UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO DE FIN DE GRADO

CREACIÓN DE MODELOS DE UN CONVERTIDOR REDUCTOR PARA SU SIMULACIÓN Y EMULACIÓN EN LAZO CERRADO

Bárbara Cristina Gil de Santivañes de Benito

Tutor: Alberto Sánchez González

Ponente: Ángel de Castro Martín

Septiembre 2014

A mi familia y Alberto

RESUMEN

El control digital aplicado a convertidores conmutados de potencia ha tenido un gran crecimiento durante la última década. En la literatura científica reciente se pueden encontrar numerosos sistemas digitales que mejoran las prestaciones o añaden características nuevas respecto a los tradicionales reguladores analógicos.

En cualquier ámbito de la ingeniería, pero aún más acusado en la electrónica de potencia, los reguladores diseñados deben ser simulados y probados antes de ser llevados a producción. Eso es debido a la gran cantidad de energía que administran y podría ocasionar grandes daños en caso de mal funcionamiento del sistema. Sin embargo, la simulación de un regulador digital para un convertidor de potencia no es trivial, ya que el regulador es digital mientras que el convertidor es un dispositivo analógico. Se han propuesto varias formas para la simulación conjunta de elementos digitales y analógicos, aunque cada técnica tiene sus ventajas y sus inconvenientes.

Este trabajo de fin de grado se basa en el diseño e implementación de un modelo digital de un convertidor reductor para su simulación conjunta con un regulador digital. La aplicación de este convertidor reductor es la carga de baterías de vehículos, la cual tiene interés industrial, pues las baterías deben someterse a varios ciclos de carga y descarga antes de su comercialización. El modelo propuesto será implementado en el lenguaje VHDL (*VHSIC Hardware Description Language*) y contará con varias versiones utilizando aritméticas diferentes, ya que la elección correcta de la aritmética es fundamental para que el modelo sea preciso.

En particular, se ha implementado una versión del modelo usando coma flotante de 64 bits, con el tipo de datos *real* de VHDL, y tres versiones en coma fija usando diferentes tamaños para las variables del modelo, con la biblioteca *sfixed*. La gran ventaja del uso de coma fija respecto al tipo de datos *real* será la posibilidad de ejecutar las pruebas dentro de una FPGA (*Field Programmable Gate Array),* emulando el sistema en vez de simularlo, lo que permitirá acelerar notablemente los resultados de la prueba del regulador junto al modelo. Este proceso de emulación de un modelo junto al regulador deseado se conoce como HIL (*Hardware-In-the-Loop*).

Por último, este trabajo fin de grado contendrá una comparación exhaustiva de los citados modelos, teniendo en cuenta factores como precisión, tiempo de simulación y emulación y recursos ocupados por los modelos en la FPGA.

Palabras clave

Hardware-In-the-Loop, simulación, emulación, electrónica de potencia, Field Programmable Gate Array.

ABSTRACT

Digital control applied to switching power converters has had a relevant growth over the last decade. In the recent scientific literature could be found a big number of digital systems that improve the benefits or they add new characteristics over traditional analog regulators.

In any engineering field, but even more marked in power electronics, each designed regulator should be simulated and tested before going to production. This is because of the high amount of energy administrated by them that could cause damage if the system works wrong. However, the digital regulator simulation for a power converter is not a trivial one since the regulator is digital and the converter is an analog device. It has been proposed different ways to do a joint simulation between digital and analogue devices, although each technique has its own advantages and disadvantages.

This work is based on the design and implementation of a digital model of a buck converter to simulate it joins a digital regulator. The application this buck converter is to charge an automotive battery, which has industrial interest because the batteries should undergo several cycles of charge and discharge before being sold. The proposed model is going to be implemented in VHDL language and it is going to feature multiple versions using different arithmetic as the correct choice of it is essential for the model to be accurate.

In particular, it has been implemented a version of the model using floating point of 64 bits, with 'real' data type from VHDL and three versions of fixed point using different sizes of the variables of the model, with 'sfixed' library. The biggest advantage of using fixed point instead of the real type will be the possibility of executing tests inside the FPGA (Field Programmable Gate Array), emulating the system and not simulating it allowing the acceleration of the results of the test regulator with de model. This process of emulation with the desired regulator is known as HIL (Hardware-In-the-Loop).

Finally, this Final project work is going to include an in-depth comparison of the above models, considering factors as accuracy, simulation and emulation time, and resources occupied by the model on the FPGA.

Index terms

Hardware-In-the-Loop, simulation, emulation, power electronics, Field Programmable Gate Array.

AGRADECIMIENTOS

Me gustaría agradecer este trabajo a todas aquellas personas que han hecho posible la llegada a este punto de inflexión tan importante en mi vida.

A mi familia, mis padres y mis hermanos, por ser un referente y un apoyo incondicional en cada decisión, simple o difícil, que he tomado en la vida. Por dar todo lo impensable y bueno que llevan dentro para sacar una sonrisa en los momentos más difíciles.

A mi tutor, Alberto, por darme la oportunidad de realizar este trabajo y prestarme la ayuda necesaria, a pesar de la distancia. Sin él no hubiera sido posible ninguna línea de esta memoria.

A mis amigos, aquellos que han pasado los años de carrera a mi lado, haciendo que hayan pasado volando gracias a su buen humor, Andrea, Alberto, Sergio Díaz, Ignacio Hernández, Eduardo Moliner, Sergio Sánchez, Roi, Tito y Javier. También a los que llevan toda la vida sin separarse de mí, Mónica, Álvaro y Marta.

Una mención especial a dos personas muy importantes que ya no están entre nosotros, pero que estoy segura de que estarán orgullosos de mí y seguirán ayudándome siempre, mi abuelo, Gonzalo, y Jota.

Por último, agradecer a la persona que más me ha apoyado durante la realización de este trabajo, durante toda la carrera y fuera de ella, Alberto, citado en los amigos, también, por ser principalmente eso, UN AMIGO. Por estar conmigo cuando lo he necesitado, por darme todo su cariño y hacer que cada día sea una razón por la que sonreír.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN Y ESTADO DEL ARTE

Un convertidor de potencia es un dispositivo capaz de variar las características de la corriente o tensión recibidas en aquellas que se deseen para un propósito determinado. Un ejemplo de convertidor de potencia básico es un divisor resistivo, el cual baja de tensión eliminando el excedente.

Desde hace muchos años, se está trabajando en convertidores de potencia conmutados, basados en el uso de elementos que cambian entre estados de conducción o aislamiento, pues el problema de muchos convertidores es su baja eficiencia.

Estos dispositivos son mucho más eficientes, ya que tienen bajas pérdidas eléctricas. Por ejemplo, un divisor resistivo, como el que se ha nombrado con anterioridad, que transforme una tensión de 20 V a una de 10 V, tiene una eficiencia aproximada del 50%, mientras que un convertidor reductor conmutado puede tenerla cercana al 95%. Además de la gran ventaja de la eficiencia, los convertidores conmutadores presentan gran fiabilidad y pueden alcanzar tamaños menores [1].

En la figura 1, extraída de [2], se muestra un esquema básico de un convertidor de potencia conmutado, el cual adapta una tensión y una corriente recibidas a la salida deseada a través del control de los interruptores observados en la figura. Existen diferentes tipos de convertidores continua-continua (*dc-dc*), entre los que se pueden destacar:

- *Boost* (elevador) es un convertidor de potencia que consigue a la salida mayor tensión que a la entrada.

- *Buck* (reductor) es un convertidor que obtiene a la salida menor tensión que a la entrada.

- *Buck-Boost* es un convertidor de potencia que puede conseguir a la salida, tanto mayor tensión que a la entrada como menor, permitiendo actuar como un convertidor elevador o como un convertidor reductor, dependiendo del control aplicado.

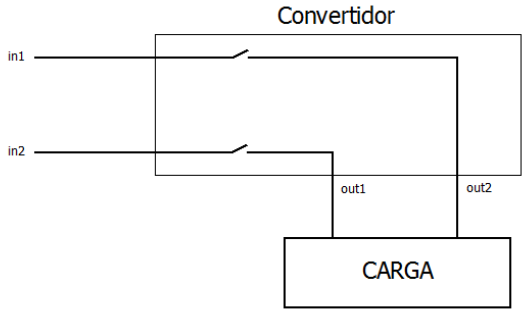

Figura 1. Esquema básico de un convertidor conmutado

Como se comentó anteriormente, años atrás los convertidores conmutados, así como los dispositivos eléctricos en general, sólo se controlaban de forma analógica, pero ahora, y cada vez de manera más habitual, se utiliza el control digital para ello. Esto es debido a que las investigaciones en este campo han crecido y se han encontrado numerosas ventajas [3] frente al control analógico. Algunas de las ventajas que presenta el control digital son:

- **Reprogramación.** Los dispositivos digitales pueden, habitualmente, ser reprogramados. Esto es un gran avance ya que se pueden realizar las modificaciones necesarias sobre el control sin necesidad de realizar modificaciones *hardware* del regulador.
- **Tiempo de diseño.** Gracias a la ventaja de reprogramación, el tiempo de diseño de un control digital, disminuye, pues cualquier cambio puede ser realizado al instante. Esta ventaja es especialmente significativa en las etapas de pruebas experimentales del regulador.
- **Fiabilidad.** El control digital en muchos casos es más fiable que el control analógico. Esto es debido a que se necesitan menos componentes, ya que éstos se incluyen en la programación, y el riesgo de fallo es menor.
- **Integración.** Un sistema analógico es difícil de integrarlo en un sistema de mayores dimensiones, pues necesitaría grandes adaptaciones según dónde quisiésemos incluirlo. En cambio, lo único que se requiere para integrar uno digital en un sistema de dimensionalidad mayor, es que éste tenga capacidad y recursos para cada una de las partes del pequeño.

Por otro lado, es cierto que el control digital presenta ciertos inconvenientes, haciendo que sigan utilizándose controles analógicos en numerosas aplicaciones. Algunos de los inconvenientes del control digital son:

- **Coste.** En un principio y todavía sigue pasando, la realización de un control digital implicaba un coste superior al analógico, ya que los elementos utilizados para este último, son muy baratos. Pero, con el gran crecimiento de lo digital, ya se pueden encontrar algunos controles que permiten competir con el precio global de lo analógico.

- **Limitación del número de bits.** En el control digital hay que lidiar con la limitación del número de bits. Esto es un problema, ya que elimina resolución de los datos. En el caso analógico esto no sucede, debido a que puede tomar infinitos valores. Esta última afirmación, no es del todo cierta, pues no es lo mismo la teoría que la práctica, donde algunos factores pueden llevar a lo analógico a valores no tan precisos como debería.
- **Conversión.** Cuando se realiza un control digital, se trabaja con variables analógicas, lo que lleva a tener que digitalizarlas para poder utilizarlas. Es por eso por lo que aquí se encuentra otro inconveniente. Se necesitan habitualmente convertidores AC/DC y DC/AC para ello.

Aún con todos estos inconvenientes, las ventajas están muy por encima y es lo que lleva a seguir investigando y desarrollando controles digitales. Esto no quiere decir, que el control analógico haya desaparecido, pues, por ejemplo, en el caso de trabajos sencillos todavía éste se sigue prefiriendo. Pero la realidad es, que cada vez se pueden encontrar más convertidores digitales tanto en ámbitos de investigación como comerciales.

1.1 ESTADO DEL ARTE DE LA SIMULACIÓN

Antes de probar un regulador con un convertidor real, es necesario realizar simulaciones. Esta etapa de pruebas es necesaria en cualquier producto de ingeniería, pero más en el caso de los convertidores de potencia, ya que éstos manejan gran cantidad de energía y, en caso de fallo, pueden ocasionar elevadas pérdidas, tanto por el coste del producto como por daños humanos. Por tanto, es indudable la importancia de una etapa de pruebas exhaustiva.

La dificultad que presenta la simulación del control digital, comparándola con la de un sistema analógico, es que se trata de un sistema en el que la planta (convertidor sobre el que se quiere realizar el control) es analógica, pues se compone de resistencias, condensadores, bobinas, mientras que el control es digital. Luego, es un sistema mixto analógico-digital, tal y como se representa en la figura 2. No existen muchos simuladores que permitan realizar este tipo de simulación mixta, y los que lo soportan son muy lentos. Un ejemplo es SystemVision [4]. Este tipo de simuladores permiten simular un esquemático tipo SPICE junto a un regulador escrito en un lenguaje de descripción *hardware*, tal como VHDL o Verilog. Como alternativa, algunas propuestas incluyen utilizar dos simuladores [5], uno para la parte analógica y otro para la digital. Esto conlleva un trabajo adicional, pues para conectar ambas aplicaciones es necesario implementar código específico por el usuario.

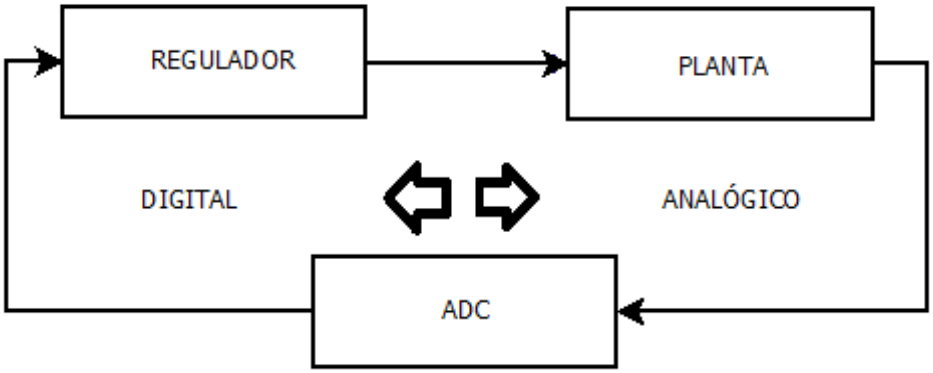

Figura 2. Esquema analógica-digital de un convertidor de potencia

Otra posibilidad, siendo la solución que se utilizará en este trabajo, es implementar un modelo del convertidor de potencia en lenguaje HDL, como se describe en [6]. De esa forma, se podrá realizar una simulación completamente digital. Por simplicidad de codificación, es bastante probable que el modelo HDL de la planta sea menos preciso que el usado en una simulación SPICE. Sin embargo, es importante remarcar que el objetivo propuesto es comprobar si el regulador contiene algún fallo de codificación HDL, tales como errores aritméticos, errores al realizar segmentación *hardware*, interfaz con el usuario o ADCs (Analogue to Digital Converter), etc. Otras pruebas importantes, tales como la estabilidad del regulador y la ganancia de sistema en lazo cerrado, se habrán realizado en la etapa de diseño del regulador, la cuales puede realizarse fácilmente con herramientas tipo SPICE, Matlab Sisotool, Simulink, etc. Por tanto, los modelos HDL se centran en probar un regulador realizado para comprobar si la implementación *hardware* es correcta.

Una gran ventaja de la codificación de un modelo HDL de la planta analógica, además de poder realizar una simulación conjunta, es que permite la emulación en *hardware,* es decir, la ejecución de la prueba utilizando dispositivos reprogramables. Este proceso de emulación de un modelo junto al regulador deseado se conoce como HIL (*Hardware-In-the-Loop*). Como se podrá ver en el capítulo 3, la emulación aporta una gran aceleración en la etapa de pruebas a costa del esfuerzo de síntesis de *hardware*, descarga en el dispositivo y recolección de datos.

1.2 OBJETIVOS Y MOTIVACIÓN

Este trabajo de fin de grado se basa en la realización de varios modelos de un convertidor reductor para su simulación y emulación en lazo cerrado. La aplicación del convertidor reductor es la formación de baterías de vehículos. Antes de poder comercializarse una batería, ésta debe someterse a varios ciclos de carga-descarga para llegar a un estado óptimo de uso. En particular, en este proyecto se modelará el circuito de carga del formador de baterías. Dicho formador permitirá la carga de 16 baterías de forma simultánea, adquiriendo la energía de una toma trifásica.

Cualquier regulador que se quiera probar en un convertidor real, primero debe ser probado en simulación, como se ha explicado, ya que si no es así, pueden ocurrir diversos daños. Debido a esto, se realizarán distintos modelos de un convertidor reductor, para que puedan ser utilizados para la simulación de reguladores. Sin embargo, no es trivial la decisión sobre qué tipo de aritmética utilizar para el modelado de la planta. De forma general, podemos distinguir aritmética en coma flotante y en coma fija.

Debido a ello, se van a presentar dos modelos del convertidor reductor con sendas aritméticas. Con dichos modelos, se podrán probar los reguladores que se deseen utilizar y poder tener gran certeza del correcto funcionamiento cuando el regulador se conecte a un convertidor real.

Con este proyecto, se tratará de demostrar las ventajas e inconvenientes entre el modelado de la planta con aritmética en coma flotante y en coma fija. Para la implementación de estos modelos, se utilizará el lenguaje VHDL, aunque se podría haber utilizado otros lenguajes HDL como Verilog.

La coma flotante es simple y cómoda de usar, mientras que la coma fija requiere un proceso más manual, en el que hay que decidir el ancho de cada señal. En VHDL y usando coma flotante es posible la utilización de la aritmética *real*, propuesta para este trabajo. Esta aritmética es muy fácil de utilizar, pero su uso está limitado al ámbito de la simulación, por lo que no se puede sintetizar, es decir, no puede ser implementada en *hardware*. Es cierto que hay implementaciones sintetizables de coma flotante como la biblioteca *float*, del paquete *VHDL2008*. Sin embargo, los diseños utilizando esta biblioteca son muy lentos en síntesis y ejecución respecto a una implementación en coma fija [6].

Por eso se propone, también, el modelado del convertidor en coma fija, que acelerará la ejecución de manera considerable. Para que este último modelado se realice de forma más sencilla, se utilizará la biblioteca *sfixed*, también del paquete VHDL2008. Esta biblioteca facilita la definición de variables en coma fija, así como las operaciones aritméticas entre ellas. Además, permite tanto ser sintetizada como emulada.

Por último y como se ha comentado anteriormente, el uso de coma fija requiere la decisión sobre qué ancho deben tener las señales HDL para que las operaciones aritméticas tengan suficiente resolución. Por ello se elaborarán tres modelos similares en coma fija, en los que variará el número de bits de las entradas del modelo. Los anchos de buses utilizados serán de 6, 12 y 16 bits.

En el capítulo 3 se podrá encontrar una comparativa entre todos los modelos propuestos. Esta comparativa incluirá la precisión de los modelos, tiempos de simulación, emulación, área ocupada, etc.

Este trabajo de fin de grado tiene la siguiente estructura de memoria:

- El primer capítulo trata la introducción, el estado del arte de la simulación y la motivación y objetivos de este trabajo.

- El segundo capítulo se basa en el modelado del convertidor reductor, el desarrollo del circuito de éste, consiguiendo las ecuaciones en diferencias para poder obtener

las señales deseadas (corriente de la bobina y tensión de salida) para ser estudiadas. Además, se presentan los dos formatos en los que ha sido programado el reductor, coma flotante y coma fija.

- El tercer capítulo presenta un análisis exhaustivo de los cuatro modelos propuestos (uno en coma flotante y tres en coma fija), considerando parámetros como la precisión, tiempo de simulación, posibilidad y tiempo de emulación y recursos *hardware* necesarios en caso de la emulación.

- En el último capítulo, veremos las conclusiones finales del trabajo de fin de grado.

- Por último, el Anexo I presenta un glosario de abreviaturas utilizadas en este documento, y el Anexo II recoge un listado de códigos usados para la simulación y emulación HIL.

2. MODELADO DE UN CONVERTIDOR REDUCTOR

En este capítulo se exponen los dos modelos que se utilizarán para la realización del convertidor reductor y su simulación con el regulador. El lenguaje utilizado será VHDL, describiéndolos usando señales de tipo coma flotante y coma fija.

2.1 MODELO DE LA PLANTA

El convertidor reductor está compuesto por tres elementos pasivos, una bobina, un condensador y una resistencia, además de por un diodo y un interruptor, como se ve en la figura 3.

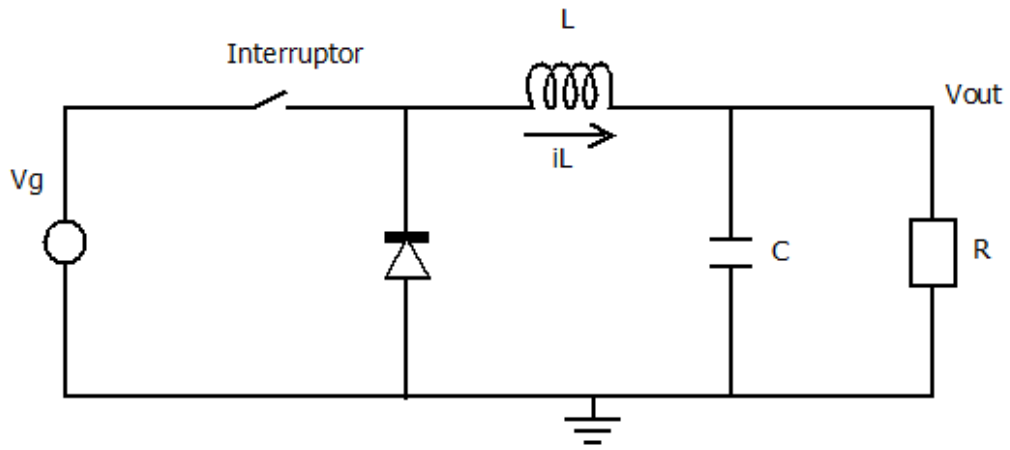

Figura 3. Esquemático de un convertidor reductor

Dado que el interruptor es un elemento activo controlable por el regulador, se distinguirán dos casos, según dicho interruptor esté cerrado o abierto. Si estuviera cerrado, se obtendría el esquema eléctrico de la figura 4.

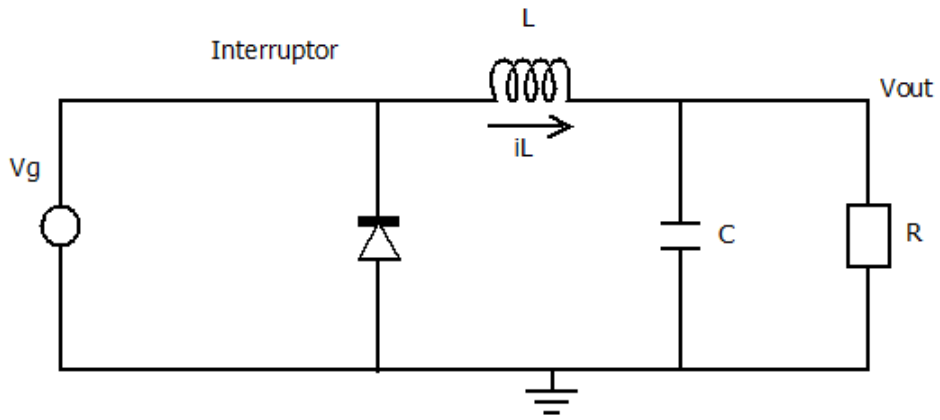

Figura 4. Esquemático de un convertidor reductor cuando el interruptor está cerrado

Cuando el interruptor está cerrado, la corriente de entrada circula a través de la bobina y, por tanto, a través del condensador y de la resistencia de salida, que modela el conjunto de baterías a formar. En este caso, el diodo no toma parte en el circuito, ya que está polarizado de forma inversa y no conduce.

Un modelo puede ser caracterizado por la evolución de sus variables de estado, que en un convertidor reductor son la corriente de entrada (i_L), y la tensión de salida (v_{out}), definidas respectivamente por la bobina y por el condensador. Por ello, deben considerarse las ecuaciones características de una bobina y de un condensador para poder definir las variables de estado:

$$v_L = L * \frac{di_L}{dt} \tag{2.1}$$

Para realizar el modelo discreto de la planta, se procede a la transformación de la ecuación (2.1) en una ecuación en diferencias en función del ciclo de integración k:

$$i_L(k) = i_L(k-1) + v_L * \frac{\Delta t}{L} \tag{2.2}$$

Donde $i_L(k)$ es la corriente que atraviesa la bobina en el instante de tiempo k , v_L es la tensión en la bobina, Δt es el tiempo de integración, es decir, cada cuánto se actualizarán las variables de estado, y L es la inductancia de la bobina.

Como puede verse en la figura 4, la tensión aplicada en los bornes de la bobina viene definida por la ecuación $v_L = v_g - v_{out}$, por lo que la ecuación (2.2) puede ser rescrita como:

$$i_L(k) = i_L(k-1) + \left(v_g - v_{out}\right) * \frac{\Delta t}{L} \tag{2.3}$$

En cada ciclo de integración, la corriente de entrada se recalculará teniendo en cuenta la ecuación (2.3). Por otro lado, la tensión de salida puede ser extraída a través de la ecuación característica del condensador:

$$i_C = C * \frac{dv_{out}}{dt} \tag{2.4}$$

Nuevamente, como en (2.1), la ecuación (2.4) puede ser transformada en una ecuación en diferencias, obteniendo:

$$v_{out}(k) = v_{out}(k-1) + i_C * \frac{\Delta t}{C} \tag{2.5}$$

Definiendo $v_{out}(k)$ como la tensión que se obtiene a la salida en el instante k, i_C es la corriente que atraviesa el condensador, que depende de la carga que se conecta a la salida, y C la capacidad del condensador.

Las baterías pueden modelarse como una carga resistiva con baja resistencia. Por tanto, se determina que para este caso, $i_C = i_L - i_R$ y se sustituye i_C en la ecuación (2.5), obteniendo:

$$v_{out}(k) = v_{out}(k-1) + (i_L - i_R) * \frac{\Delta t}{C} \tag{2.6}$$

Llegados a este punto, se han conseguido las dos ecuaciones que determinarán las variables de estado en el caso en el que el interruptor está cerrado. Sin embargo, el regulador abrirá periódicamente el interruptor, por lo que es necesario definir la evolución de las variables de estado en ese estado. En el caso de interruptor abierto, el convertidor queda como el mostrado en la figura 5. La fuente de entrada no aporta energía a la salida del convertidor, por tanto, la corriente que atraviesa la bobina se reducirá paulatinamente hasta hacerse nula. En el caso en el que la corriente de entrada no sea

nula, el diodo estará polarizado en directa y, por tanto, conducirá. Sin embargo, puede llegar un momento en el que la corriente se anule.

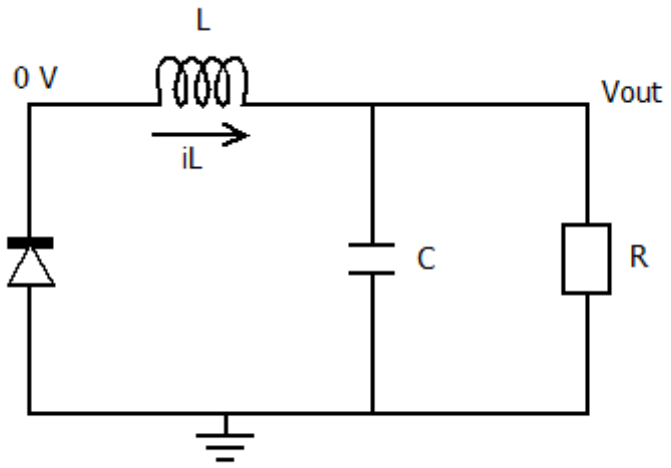

Figura 5. Esquemático de un convertidor reductor en el estado de CCM

Por tanto, podemos distinguir dos casos cuando el interruptor está abierto: llamados CCM y DCM.

2.1.1 CCM (CONTINUOUS CONDUCTION MODE)

Este caso se muestra en la figura 5. La corriente que atraviesa la bobina no llega a ser nula, pues aunque no haya ningún elemento que haga que la bobina acumule energía, ya que el interruptor está abierto y la tensión de entrada no es entregada al circuito, existe una energía acumulada de cuando el interruptor estaba cerrado.

Al estar el interruptor abierto, entre los bornes de la bobina habrá una tensión definida como $v_L = 0 - v_{out}$. Por tanto, sustituyendo el nuevo valor de v_L en la ecuación (2.2) se obtiene el valor de la corriente en CCM:

$$i_L(k) = i_L(k-1) - v_{out} * \frac{\Delta t}{L} \tag{2.7}$$

En el caso del cálculo de $v_{out}(k)$, se cumple la misma ecuación que en el caso del interruptor cerrado, ya que la corriente que pasa por la bobina, i_L, todavía no ha llegado a hacerse cero:

$$v_{out}(k) = v_{out}(k-1) + (i_L - i_R) * \frac{\Delta t}{C} \tag{2.8}$$

2.1.2 DCM (DISCONTINUOUS CONDUCTION MODE)

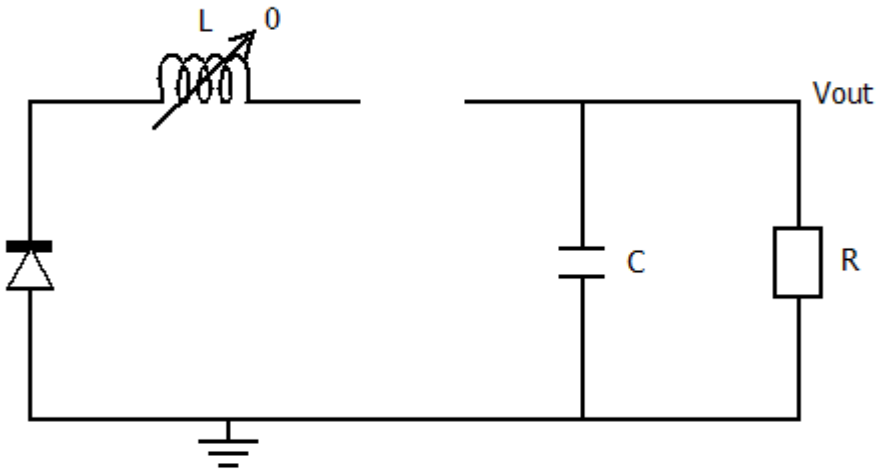

Figura 6. Esquemático de un convertidor reductor en estado de DCM

El modo discontinuo, mostrado en la figura 6, se caracteriza porque la corriente i_L es nula tras haber pasado por el modo continuo. Por tanto, la ecuación de la corriente que atraviesa la bobina es:

$$i_L(k) = 0 \tag{2.9}$$

Y, en el caso de $v_{out}(k)$, ahora se cumple $i_C = i_L - i_R = 0 - i_R$, luego:

$$v_{out}(k) = v_{out}(k-1) - i_R * \frac{\Delta t}{C} \tag{2.10}$$

2.2 RÉGIMEN PERMANENTE

Aunque desde el punto de vista del modelado no es importante, se va a detallar qué tensión de salida se obtendrá en régimen permanente cuando se aplica un determinado ciclo de trabajo al interruptor.

Si se considera que el convertidor opera en condiciones estacionarias, la cantidad de energía almacenada en la bobina, debe ser la misma al principio y al final del ciclo completo de conmutación. Esto es lo que se denomina como régimen permanente. Así pues, la corriente de la bobina tiene que ser idéntica al principio y al final del ciclo de conmutación, y responderá a la ecuación (2.11), que posteriormente se desarrollará dando lugar a las ecuaciones (2.12) y (2.13):

$$\Delta i_{Lon} + \Delta i_{Loff} = 0 \tag{2.11}$$

$$(v - v_{out}) * \frac{dt_{on}}{L} + (-v_{out}) * \frac{dt_{off}}{L} = 0 \tag{2.12}$$

$$\left(v_g - v_{out}\right) * dt_{on} + (-v_{out}) * dt_{off} = 0 \tag{2.13}$$

Se tendrá en cuenta que $dt_{off} = (1 - dt_{on})$, ya que se considera que el ciclo de trabajo está normalizado a la unidad, y se renombrará dt_{on} como d. Por tanto, la ecuación (2.13) se desarrollará como:

$$\left(v_g - v_{out}\right) * d + (-v_{out}) * (1 - d) = 0 \tag{2.14}$$

$$v_g * d - v_{out} * d - v_{out} + v_{out} * d = 0 \tag{2.15}$$

$$v_g * d = v_{out} \tag{2.16}$$

$$v_{out} = v_g * d \tag{2.17}$$

Por tanto, en régimen permanente, la tensión de salida será proporcional a la tensión de entrada y regulada según el ciclo de trabajo *d.* Como se puede observar, la tensión de salida siempre será menor a la tensión de entrada, ya que el ciclo de trabajo puede variar entre 0 y 1.

2.3 IMPLEMENTACIÓN DEL MODELO

En este apartado se van a presentar diferentes maneras de realizar el modelo del convertidor reductor en VHDL y simularlo con un regulador digital. Para ello, se proponen dos modelos a diseñar, uno que utilice señales en coma flotante (*real*), y otro en coma fija (*QX.Y*), usando la biblioteca *sfixed.*

La ventaja de la utilización de la coma flotante, en particular el tipo de "datos" *real*, es que ésta la mayoría de herramientas de simulación la aceptan. Sin embargo, la aritmética *real* no puede ser sintetizada, como se ha explicado anteriormente. En cambio, el modelo en coma fija, permitirá la simulación, síntesis y emulación del diseño, utilizando menos *hardware* y aumentando la velocidad. A pesar de que se requiere un mayor esfuerzo a la hora de ser implementado por parte del diseñador, el convertidor solo se realizará una vez, a diferencia del regulador, que dependiendo de cada prueba, deberá ser modificado.

2.2.1 MODELO REAL

La ventaja de realizar un modelo con un tipo de señales *real* es que está soportado por la mayoría de simuladores y que su implementación es muy sencilla. Sin embargo, su mayor desventaja es que no puede ser sintetizado, es decir, no puede ser traducido a *hardware* real.

En el caso del modelado del convertidor en formato de tipo *real*, se trata de un circuito síncrono, en el que en cada ciclo de reloj (*rising_edge* en VHDL) se realiza una nueva integración, lo que da lugar al cálculo de las variables deseadas, corriente en la bobina y tensión de salida, con el tiempo de integración que se defina con anterioridad.

Para realizar el modelo del convertidor de potencia *Buck* en HDL, se utilizarán las ecuaciones en diferencias (2.3) y (2.6), para el caso del interruptor cerrado y (2.7), (2.8), (2.9) y (2.10) para el caso del interruptor abierto., sin realizar modificaciones, ya que son simples y el tipo *real* acepta esta simplicidad.

Seguidamente, se muestra en la figura 7 el esquema básico del convertidor reductor para el caso del modelado en formato *real*. Como se puede observar, es un esquemático sencillo, basado en dos *flip-flops*, que permiten el registro de las variables de entorno en cada ciclo de reloj, además de la utilización de cuatro multiplexores, cuatro multiplicadores y cuatro sumadores (dos actuando, propiamente, como sumadores y otros dos como sumadores restadores).

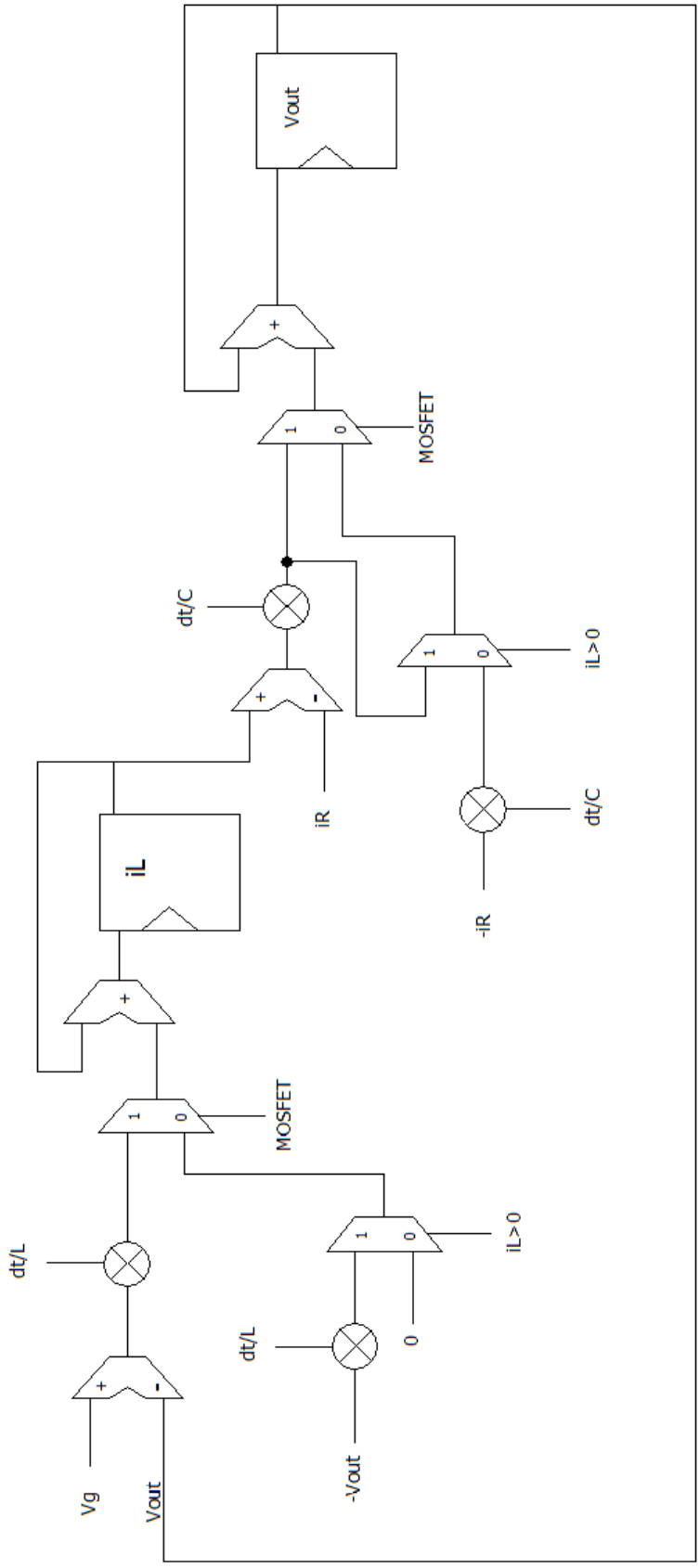

Figura 7. Esquema básico del convertidor reductor para su modelado en formato real

Por un lado, en la parte inferior de la figura 7, desde la salida del registro en el que está escrito i_L, hacia la parte inferior, se presenta el cálculo de la corriente que atraviesa la bobina, siguiendo las ecuaciones en diferencias que se describieron en el apartado 2.1.

Como se puede ver, la señal que sale de este registro es la que se utiliza para actualizar la propia corriente de la bobina, añadiendo ese valor al nuevo en cada ciclo de reloj. La actualización viene determinada por los multiplexores, que se encargan de comprobar en qué estado se encuentra el convertidor, es decir, si el interruptor está cerrado o abierto, y si es este último caso, diferenciando entre CCM o DCM. Además, el cálculo de i_L depende, también, de la tensión de entrada y de la de salida, que dependiendo del modo en el que se encuentre el circuito, se utilizará la resta de éstas o el valor en negativo de la tensión de salida. Estos valores son multiplicados por una escala, que está determinada por el tiempo de integración y el valor de la bobina con la que se trabaja, ya que es lo que indican las ecuaciones en diferencia.

Por otro lado, en la parte superior de la figura 7, se halla el cálculo de la tensión de salida. Como en el caso de la corriente de la bobina, la señal que sale del registro con el nombre v_{out}, es la que se realimenta y se añade al nuevo valor del voltaje, para así ir actualizándose en cada ciclo de reloj.

El nuevo valor que hace incrementar la tensión de salida viene controlado por otros dos multiplexores, encargados del mismo control que en el caso anterior. Pero, ahora, el incremento que añadimos, viene de la resta de la corriente de la bobina, que sale del registro anterior, es decir i_L, con la corriente de la carga (i_R), que se ha conectado al *Buck*. O, si estamos en DCM, solo de la corriente de la carga ($-i_R$ en el esquema). Como en el cálculo de i_L, es necesario multiplicar las señales por el tiempo de integración dividido, en este caso, por el valor del condensador.

Ahora se procederá a traspasar esta explicación y el esquema de la figura 7 al lenguaje en el que se trabajará, VHDL, pues es así como luego se realizará la simulación y pruebas del trabajo de fin de grado. Para la implementación sería posible realizar todo el esquema en un mismo proceso, pero se ha decidido hacerlo en dos por simplicidad. Por lo tanto, el convertidor queda dividido en dos procesos. El primero, mostrado en la figura 8, es el encargado del cálculo de los incrementos de las variables que se estudian, corriente de entrada y tensión de salida (i_L, v_{out}), reconocidas en el código como il_aux y $vout\_aux$, respectivamente. El segundo proceso, figura 9, realiza la actualización de las dos variables que se calculan en el proceso anterior, para cada ciclo de reloj (*rising_edge*). Esta actualización, es la acumulación del valor de la señal en el estado actual con el estado anterior o estados anteriores, pues es lo que se ha determinado en el desarrollo de las ecuaciones en diferencias

Aparecen en el primer proceso, además de las citadas, tres variables más:

- dt, es el tiempo de integración (Δt), tiempo que se integra en cada ciclo de reloj. Para que la integración sea precisa, el tiempo tiene que ser pequeño, para así hacer incrementos pequeños. Para este caso se ha elegido una frecuencia de 100 MHz, lo que corresponde a un tiempo de integración de 10 ns.

- L, es la inductancia de la bobina. El valor que le corresponde es 2 mH.

- C, siendo la capacidad del condensador. Para esta implementación tiene un valor de 35 µF.

```
--Cálculo de las señales iL y vout
process(MOSFET,Vg,v_out,il,Ir)
begin
   --Interruptor cerrado
   if Mosfet = '1' then
      il_aux <= (Vg-v_out)*(dt/L);
      vout_aux <= (il-Ir)*(dt/C);

   --Interruptor abierto
   else
      if il <= 0.0 then --Circuito en DCM (Discontinuous Conduction Mode)
         il_aux <= 0.0;
         vout_aux <= -(Ir*(dt/C));

      else  --Circuito en CCM (Continuous Conduction Mode)
         il_aux <= -(v_out*(dt/L));
         vout_aux <= (il-Ir)*(dt/C);

      end if;
   end if;
end process;
```

Figura 8. Cálculo de las señales. Modelo *real* del convertidor reductor

Al estar realizando un circuito síncrono, el modelo tiene habilitada una función de *reset* para poder iniciar la simulación desde un punto de partida conocido. Esto se realiza en el segundo proceso, cuyo código se presenta en la figura 9. Las nuevas señales, ya habiendo sido actualizadas debidamente, reciben el nombre en el diseño en VHDL de *il* y *v_out*.

```
--Actualización de las señales iL y vout en cada ciclo de reloj
process(Clk, Reset)
begin
   if Reset = '1' then
      il <= il_inicial;
      v_out <= vout_inicial;

   elsif rising_edge(Clk) then
      il <= il + il_aux;
      v_out <= v_out + vout_aux;

   end if;
end process;
```

Figura 9. Actualización de las señales. Modelo *real* del convertidor reductor

Como se puede observar, la realización del modelo del convertidor reductor, cuando se trata de la implementación con variables de tipo *real,* se basa en la simple transcripción de las ecuaciones en diferencias descritas en el apartado 2.1 pasadas a lenguaje VHDL. Luego, el tiempo de diseño de este modelo es muy reducido. No pasará lo mismo en el caso del modelo en coma fija, con variables de tipo *sfixed*, como se verá en el apartado 2.2.2.

2.2.2 MODELO COMA FIJA

Al realizar un modelo con señales en coma fija, como se ha dicho antes, existe la posibilidad de que pueda ser implementado en *hardware* real, por lo que se puede realizar una emulación en vez de simulación software. La emulación se caracteriza por ser mucho más veloz que la simulación, reduciendo notablemente el tiempo de pruebas del convertidor de potencia.

El problema que se puede encontrar con este tipo de notación, es que hay que tener en cuenta muchos factores a la hora de implementarlo en lenguaje HDL. Esto es debido a que toda señal debe tener su formato individual, teniendo en cuenta su tamaño y que no se desborde, ya que se realizan operaciones aritméticas con ellas, que aumentan su tamaño, y hay que haberlo previsto con anterioridad. Además, es necesario que la resolución sea, al menos, suficiente para mostrar los resultados deseados.

Puesto que en este apartado se propone realizar un modelo del convertidor en coma fija, requiriendo un esfuerzo añadido a cambio de conseguir una mayor velocidad de emulación, se pueden realizar una serie de modificaciones matemáticas en las ecuaciones en diferencias, desarrolladas en el apartado 2.1, que ayuden a reducir la complejidad del *hardware* del modelo resultante. De esta forma, el tiempo de diseño aumentará, pero ganaremos en que la emulación será más rápida. Así, se conseguirá un modelado del convertidor reductor lo más óptimo posible, llegando a mejorar el resultado final al que va destinado este trabajo.

La modificación principal que se propone, se basa en no realizar las multiplicaciones que impliquen a $\frac{dt}{L}$ y $\frac{dt}{C}$ [6], ya que al ser multiplicaciones que se realizan en cada ciclo, hacen que se pierda mucha velocidad. Luego, las nuevas ecuaciones quedarían de la siguiente manera:

$$i_L^*(k) = i_L^*(k-1) + v_L \tag{2.19}$$

$$v_{out}^*(k) = v_{out}^*(k-1) + i_C \tag{2.20}$$

Donde $i_L^*(k) = i_L * \frac{L}{\Delta t}$ (2.21)

y $$v_{out}^{*}(k) = v_{out} * \frac{C}{\Delta t} \quad (2.22)$$

Pero, como se sabe, la tensión de salida no sólo depende de la corriente de carga y descarga del condensador, sino que, también, depende de la corriente de la bobina. Sin embargo, no se calcula directamente la corriente de la bobina (i_L) sino la corriente transformada, i_L^*. Por tanto, la transformada (2.21) debe aplicarse también en el cálculo de la tensión de salida, quedando ésta como:

$$v_{out}^{**}(k) = v_{out}^{**}(k-1) + i_C^* \quad (2.23)$$

Donde $$v_{out}^{**}(k) = v_{out}^{*} * \frac{L}{\Delta t} = v_{out} * \frac{C}{\Delta t} * \frac{L}{\Delta t} \quad (2.24)$$

Todas estas modificaciones, tienen que ser aplicadas a cada caso, ya esté el interruptor abierto o cerrado, pues en ellos se utilizan las multiplicaciones que están tratándose de evitar.

Por tanto, las nuevas ecuaciones que se deben implementar en VHDL son, para el interruptor cerrado (2.25) y (2.26), mientras que para el caso en el que está abierto (2.27), (2.28), (2.29) y (2.30), estando en CCM y DCM, respectivamente:

$$i_L^*(k) = i_L^*(k-1) + \left(v_g - v_{out}\right) \quad (2.25)$$

$$v_{out}^{**}(k) = v_{out}^{**}(k-1) + (i_L^* - i_R^*) \quad (2.26)$$

$$i_L^*(k) = i_L^*(k-1) - v_{out} \quad (2.27)$$

$$v_{out}^{**}(k) = v_{out}^{**}(k-1) + (i_L^* - i_R^*) * \frac{\Delta t}{C} \quad (2.28)$$

$$i_L^*(k) = 0 \quad (2.29)$$

$$v_{out}^{**}(k) = v_{out}^{**}(k-1) - i_R^* \quad (2.30)$$

La transformación (2.24), debe ser deshecha, ya que el cálculo de i_L^* requiere el valor de v_{out}. Es cierto que deshacer dicha transformación requiere una multiplicación, generando el camino crítico en el cálculo del modelo desde un punto de vista de frecuencia del modelo sintetizado. Sin embargo, el hecho de haber un camino crítico en vez dos (si no se hubieran hecho las transformaciones) hará que el diseño sintetizado sea más eficiente en cuanto a frecuencia, ya que será más fácil de optimizar.

Anteriormente, se ha mencionado el formato QX.Y de las señales, pero no se ha entrado en detalle. QX.Y es un tipo de formato numérico para representar números en coma fija. Coma fija se refiere a una notación que consiste en destinar una cantidad fija de bits para la parte entera de un valor y otra para la parte fraccionaria del mismo. El número de bits que compone la parte fraccionaria indica la posición en la que se situará la coma del número que se quiere representar.

En QX.Y, el valor que se sustituirá por la X hace referencia al número de bits de la parte entera, y el valor Y, el de la parte fraccionaria. Además, se añade un bit más que permite representar valores con signo, para indicar si el número que se transforma es positivo o negativo, pero no se recuenta ni en la X ni en la Y. Por último, decir que se utiliza complemento a 2.

Como ejemplo, un número que esté codificado en formato QX.Y, de la forma Q2.1, tendrá dos bits para la parte entera, uno para la parte decimal y uno, adicional, para el signo. Cuatro bits en total. En la tabla 1, se muestran algunos ejemplos con números en formato QX.Y. Como puede verse, un mismo número en binario, puede interpretarse de diferente manera dependiendo del formato QX.Y que se contemple. Como se ha explicado en el párrafo anterior, el valor en decimal depende de los valores que se elijan para la X y la Y.

En el caso del primer número en binario, 000110, si se quiere averiguar el valor aplicándole el formato Q3.2, pasaría a tener la coma dos posiciones a la izquierda desde el último dígito, 0001.10, y desde ahí ya se calcularía el valor en decimal teniendo en cuenta que está representado en complemento a 2.

Binario/Formato	Q3.2	Q1.4	Q2.3
000110	1.5	0.375	0.75
110011	-3.25	-0.8125	-1.625
010000	4	1	2

Tabla 1. Ejemplos de valores en binario representados en distintos formatos QX.Y

Una vez explicado el formato de representación usado en este trabajo, se va a proceder a explicar la implementación del modelo en coma fija. En la figura 10, se muestra el modelo del convertidor *Buck* en coma fija, con los valores de las señales en formato QX.Y.

En la parte inferior de la figura 10, como el caso del modelo en real, desde la salida del registro en el que está escrito i_L^* hacia abajo, se presenta el cálculo de la corriente que atraviesa la bobina, pero ahora siguiendo las ecuaciones en diferencias optimizadas para su implementación en *hardware*, como se describió al principio de este apartado.

Un factor a tener en cuenta en el modelo es el tamaño de las entradas, ya que éste va a afectar enormemente a la resolución del modelo. Como se puede ver, las entradas del modelo son la tensión de entrada, y la corriente de la carga. Se ha optado por elegir un ancho de entradas de 12 bits, que se supone que es suficiente. Este valor elegido puede parecer trivial, pero la realidad es que en aplicaciones de potencia muchas veces se utilizan ADCs de 8 y 10 bits con resultados satisfactorios. Por tanto, modelar la tensión de entrada y la corriente de salida con 12 bits a priori debe ser suficiente. En cualquier caso, es razonable cuestionar la elección de este ancho de entradas, por lo que en el capítulo de resultados se encontrará una comparativa entre modelos con 6, 12 y 16 bits en las entradas. Por simplicidad, la explicación del modelo en coma fija se centrará únicamente en la versión con 12 bits.

Como se puede observar, la señal que sale del registro i_L^* que se encuentra en formato Q25.2, es la que se utiliza para actualizar la propia corriente de la bobina. El número de bits de esta señal es bastante grande, 28 bits; esto es debido a que es una variable de estado y necesitamos mucha resolución, pues tiene que integrar los pequeños incrementos a la vez que almacena el valor instantáneo de dicha variable. En el caso propuesto, por ejemplo, la corriente alcanzará valores en torno a los 90 A, mientras que el incremento de ésta, durante un ciclo de reloj, puede ser alrededor de los microamperios. Es por esta gran diferencia, entre el valor instantáneo y el valor incremental, por lo que la señal debe tener un número tan grande de bits, pues necesita almacenar ambos valores.

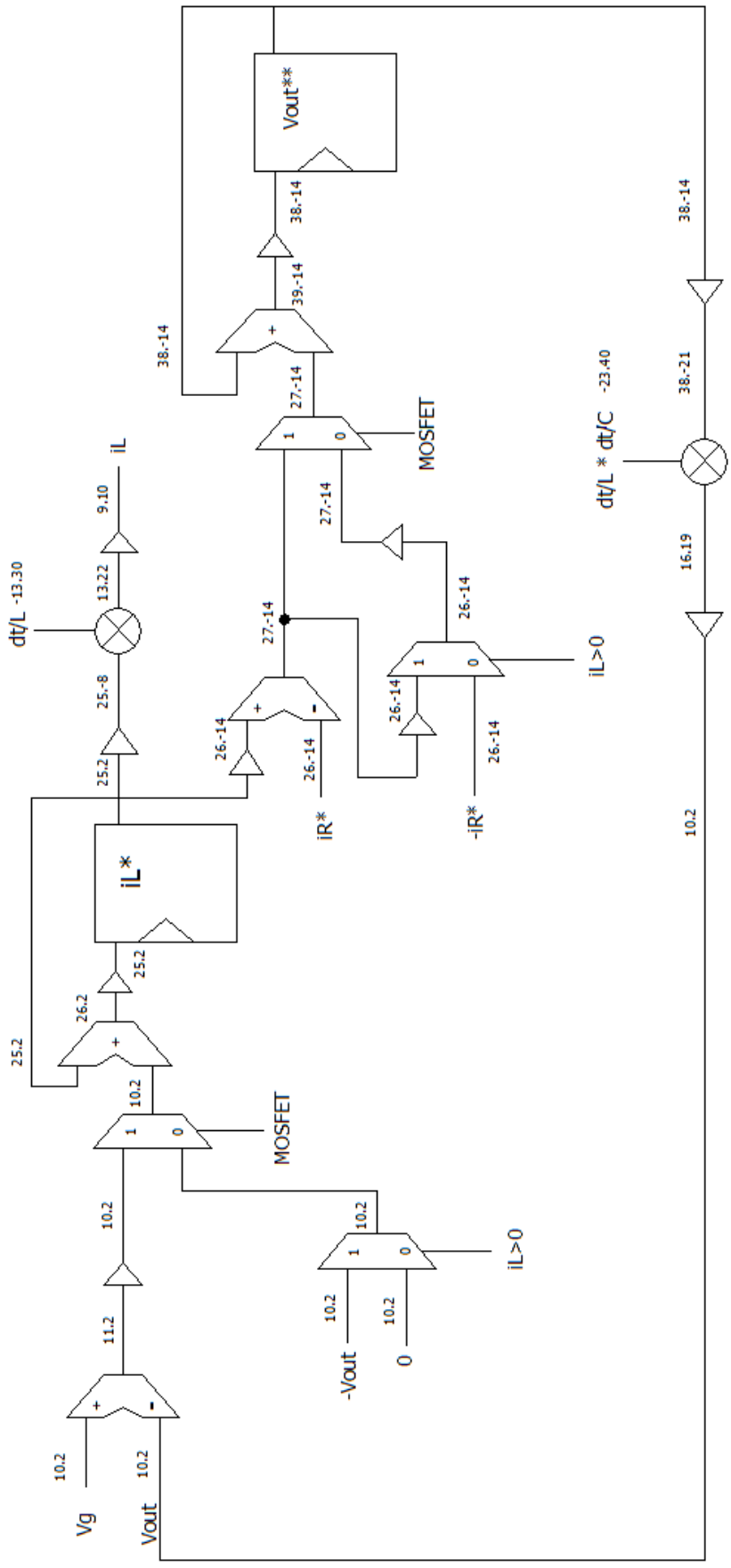

Figura 10. Esquema básico del convertidor reductor para el caso del modelado en formato QX.Y

Además, el cálculo de i_L^* depende, como se dijo antes, de la tensión de entrada y de la de salida, con un formato de Q10.2. El número de bits, en este caso, no es necesario que sea grande, pues se trata de variables de entrada y salida, donde no se realiza ninguna integración, sino que con ellas se representa un valor instantáneo. Luego, no es necesaria tanta precisión.

Por otro lado, en la parte superior derecha de la figura 10, se halla el cálculo de la tensión de salida. Como en el caso de la corriente de la bobina, la señal que sale del registro con el nombre v_{out}^{**}, con un formato Q38.-14, es la que realimenta y se añade al nuevo valor del voltaje, para así ir actualizándose en cada ciclo de reloj.

Al final, una vez conseguida la tensión de la salida, se vuelve a ver que tenemos el valor de v_{out}^{**}. Esto significa, que hay que deshacer los cambios, es decir, multiplicar por la escala correspondiente, en este caso $\frac{\Delta t}{L} * \frac{\Delta t}{C}$. Esto es debido, a que el valor es necesario para cerrar el circuito, ya que se trata de un lazo cerrado, y el valor de la tensión de salida es utilizado para el cálculo de la corriente de la bobina, y se necesita en el estado normal, sin escalas.

Como se puede ver en la figura 10, el modelo no necesita deshacer la transformación de i_L^* para su funcionamiento, sino que dicha transformación se ve arrastrada hacia el cálculo de la tensión de salida. Sin embargo, dependiendo de la aplicación podría ser útil conocer el valor de la corriente de entrada sin transformar. Si esto se necesitara, la salida del registro de i_L^*, se multiplicará por $\frac{\Delta t}{L}$ para obtener i_L. Esta multiplicación, en cualquier caso, no es necesario que vaya a la misma frecuencia de actualización que el modelo, por lo que no afectará a la optimización de la síntesis en *hardware*.

En la tabla 2, se muestran una serie de datos relativos a las señales más importantes del convertidor, como son el número de bits utilizados para recoger los valores de éstas, los formatos de las variables en QX.Y y su escala. Como se puede observar, el ancho de las entradas y salidas utilizadas es de 12 bits más 1 de signo. Este ancho de 12 bits ha sido elegido porque con 12 bits debería poderse representar de forma correcta tensiones y corrientes. De hecho, los ADCs utilizados en aplicaciones de potencia muchas veces tienen únicamente 8 bits y la resolución es suficiente para ellas. El valor de escala recogido es utilizado para calcular el valor real de las señales. Teniendo una señal con un formato QX.Y que tenga valor en la columna de 'Escala', para calcular el valor real a partir del literal, hay que multiplicar el valor en binario, pasado a decimal, por 2^{-Y} y por la escala. En caso de que no tenga escala, solo se realizará la multiplicación por 2^{-Y}[3].

Además, en la tabla 2, se exponen los rangos equivalentes de cada señal, para el modelo del convertidor en coma fija, además de la resolución equivalente que permiten éstas. Estos valores han sido determinados teniendo en cuenta el formato QX.Y y la escala por la que deben ser multiplicados, en el caso de las variables nuevas que se obtuvieron en el desarrollo de las ecuaciones en diferencias con las transformaciones hechas.

Señal	Número de bits	Formato	Escala	Rango equivalente	Resolución
V_g	13	Q10.2	-	-1204 a 1023.75 V	0.25
V_{out}	13	Q10.2	-	-1204 a 1023.75 V	0.25
V_{out}^*	25	Q38.-14	$\frac{dt}{L} * \frac{dt}{C}$	± 392.683 V	$2.341 \cdot 10^{-5}$
i_L	20	Q9.10	-	-512 a 511.999 A	$9.766 \cdot 10^{-4}$
i_L^*	28	Q25.2	$\frac{dt}{L}$	±167.772 A	$1.25 \cdot 10^{-6}$
i_R^*	13	Q26.-14	$\frac{dt}{L}$	-335.544 a 335.462 A	0.082

Tabla 2. Formato QX.Y de las señales para el modelo en coma fija

Como se puede observar en la tabla 2 y como se comentó previamente, prestando atención al número de bits, las variables de estado necesitan de mucha resolución, pues van integrando el modelo, es por eso por lo que el número de bits que las forman es grande, para así poder guardar valores suficientemente precisos, derivados de los pequeños incrementos sumados al valor instantáneo de la señal. En cambio, las señales que actúan como entradas o salidas y como realimentación no necesitan tanta precisión, pues sólo se utilizan para indicar el valor de la corriente y la tensión en un momento puntual, como se explicó anteriormente. El formato QX.Y dependerá del número de bits permitidos por el ADC y los multiplicadores.

Para implementar este modelo de coma fija en HDL, se ha optado por realizarlo directamente utilizando la biblioteca *sfixed* del estándar VHDL2008, ya que permite representar los datos en coma fija con signo.

Una de las ventajas de utilizar esta biblioteca es que acepta la variación en el tamaño de las variables en función de otras. Es decir, no hace falta declarar una nueva señal con cada uno de los valores QX.Y deseados para transferir el valor de una señal a otra, sino que con la función *resize*, se pueden poner las dos variables que deseadas, y el valor de la primera se copia en la señal de destino atendiendo al formato QX.Y de la segunda variable. Ayudas como la mencionada hacen que el uso de esta biblioteca facilite la labor de realizar el modelo en coma fija.

```
Vg_aux <= to_sfixed (Vg,Vg_aux);
Ir_aux <= to_sfixed (Ir,Ir_aux);

process(Mosfet,Vg_aux,v_out,il,il_as,Ir_aux,il_ajustar,ilas_ajustar)
begin
   --interruptor cerrado(SW=1)
   if Mosfet = '1' then
      il_aux <= resize(Vg_aux-v_out,il_aux);
      ilas_ajustar <= resize(il_ajustar,ilas_ajustar);
      vout_aux <= resize(ilas_ajustar-Ir_aux,vout_aux);

   else
      --DCM
      if il <= 0.0 then
         il_aux <= to_sfixed(0.0, il_aux);
         vout_aux <= resize(-Ir_aux,vout_aux);

      --CCM
      else
         il_aux <= resize(-v_out,il_aux);
         ilas_ajustar <= resize(il_ajustar,ilas_ajustar);
         vout_aux <= resize(ilas_ajustar-Ir_aux,vout_aux);

      end if;
   end if;
end process;
```

Figura 11. Cálculo de las señales. Modelo *sfixed* del convertidor reductor

En la figura 11, se presenta el código implementado para la realización del convertidor *Buck* en coma fija, utilizando la biblioteca *sfixed*.

En el modelo implementado, las entradas y salidas están declaradas como *std_logic_vector* para que el modelo sea compatible con módulos VHDL no implementados con *sfixed*. Las primeras líneas del código realizan la conversión, mediante la función *to_sfixed*, de *std_logic_vector* a *sfixed*.

En segundo lugar, la figura 11 muestra el proceso en el que se calcula el incremento en el valor de la corriente de la bobina y en la tensión de salida, distinguiendo el modo en el que se encuentra el circuito.

Como puede verse, es utilizada la función *resize*, explicada con anterioridad. Por ejemplo, realizamos la resta de *Vg_aux-v_out*, y el resultado se transforma al formato QX.Y que tiene el segundo parámetro de la función *il_aux*. Así no se declara una variable con el formato que tuviese esa resta ni otra más para ajustarla al de *il_aux*.

En la figura 12, el proceso es el de actualización de las dos variables de estado, sumando el valor anterior con el nuevo incremento. También, se emplea la función *resize*, pues facilita y simplifica el código de manera considerable.

En paralelo se procede a la multiplicación de las variables de estado por sus correspondientes escalas, como fue explicado cuando se mostró el esquema general del convertidor en QX.Y.

```
process(Clk, Reset)
begin
   if Reset = '1' then
      il_ajustar <= resize(il_inicial,il_ajustar);
      vout_ajustar <= resize(vout_inicial,vout_ajustar);

   elsif rising_edge(Clk) then

      il_ajustar <= resize(il_ajustar + il_aux,il_ajustar);
      vout_ajustar <= resize(vout_ajustar + vout_aux,vout_ajustar);


   end if;
end process;

--sacar iL=(iL*)*(dt/L)
ilMult_ajustar <= resize(il_ajustar,ilMult_ajustar);
il <= resize(ilMult_ajustar*dtL, il);

--sacar Vout=(Vout*)*(dt/L)*(dt/C)
voutMult_ajustar <= resize(vout_ajustar,voutMult_ajustar);
v_out <= resize(voutMult_ajustar*VOUTSCALE, v_out);
```

Figura 12. Actualización de las señales. Modelo *sfixed* del convertidor reductor

Se puede ver, que la realización del modelo del convertidor reductor en coma fija, utiliza las ecuaciones en diferencias optimizadas para su implementación en *hardware*, descritas al principio de este apartado, pero realizando el ajuste del tamaño de las variables, lo que aumenta el tiempo de diseño.

3. RESULTADOS

En este capítulo se procede a la comparación y presentación de los resultados de los cuatro modelos de convertidor *Buck* que se han implementado para este trabajo: el modelo de tipo *real*, y el modelo en coma fija con entradas de 6, 12 y 16 bits. Aunque lo adecuado sería probar la precisión de los cuatro modelos con un convertidor real, dicha comparación se escapa de este trabajo. Por tanto, el modelo de tipo *real* servirá como referencia, pues aunque ha sido creado nuevamente para este proyecto, en [6] se comprobó que un modelo en coma flotante en 64 bits permite modelar correctamente convertidores similares como puede ser un convertidor *boost*. Luego, como se ha dicho anteriormente, el tipo *real* se utilizará como el modelo de referencia y se comprobará la precisión de los diferentes modelos en coma fija, así como su aceleración al ser emulados.

El código, que modela el convertidor en las distintas versiones de coma flotante y coma fija, fue implementado en el entorno Xilinx ISE y las pruebas para la comparación de los resultados de los modelos se realizaron utilizando la herramienta Mentor ModelSim.

Por una parte, se realizarán pruebas de los modelos en lazo abierto, es decir, aplicando un ciclo de trabajo fijo y sin ningún tipo de control. En todas las pruebas la tensión de entrada es alterna y los ciclos de trabajos probados serán del 30% y del 50%. Las pruebas en lazo abierto y régimen permanente mostrarán la precisión de los modelos en coma fija sin que los resultados se vean alterados por un regulador.

Por otra parte, se han probado los modelos junto a un regulador que controla el nivel medio de corriente de entrada en el convertidor. Para ello se podrá observar el comportamiento dinámico con escalones en la referencia de corriente entre 13 A, 30 A y 5 A.

3.1 LAZO ABIERTO

Como se ha comentado anteriormente, un sistema de control en lazo abierto es aquel en el que no existe retroalimentación hacia el control. Este estado, provoca que el sistema no pueda realizar ninguna función de ajuste, ya que la salida no vuelve a la entrada, para poder ser recalculada. Gracias a esto, se podrá comprobar la precisión de los modelos sin ninguna perturbación externa.

3.1.1 CICLO DE TRABAJO FIJO DEL 30%

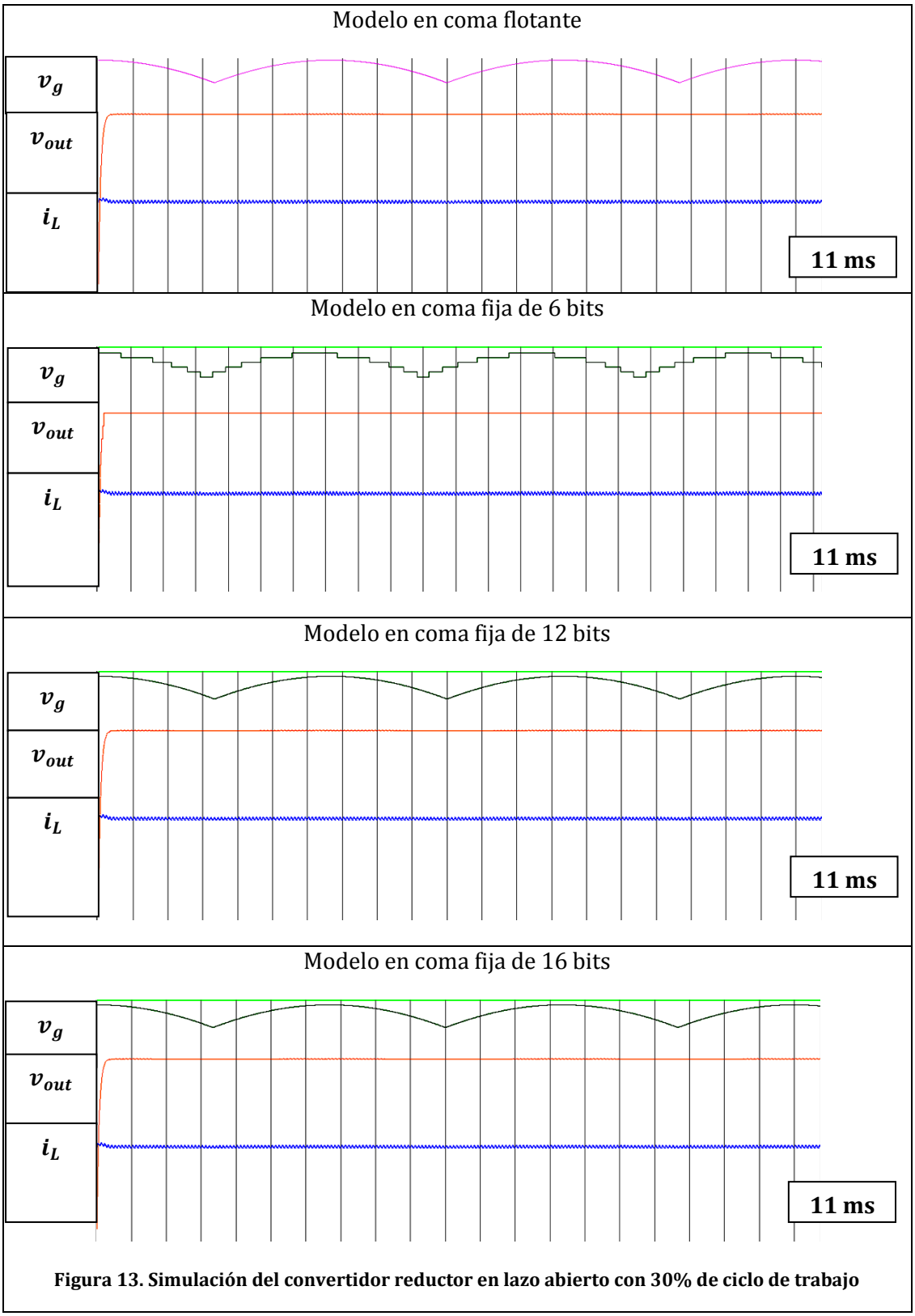

Figura 13. Simulación del convertidor reductor en lazo abierto con 30% de ciclo de trabajo

En la figura 13 se representa la medición de las variables de estado de cuatro modelos del convertidor diferentes, uno en coma flotante y tres en coma fija. Como se explicó en el apartado 2.1 de este trabajo, se pueden distinguir dos estados del convertidor: CCM y DCM. En este caso, el convertidor entra ligeramente en estado de conducción discontinua ya que la corriente llega a ser nula durante un breve instante de tiempo. En la figura 14 se ha ampliado el inicio de la simulación para que pueda ser observado que la corriente llega a ser nula en cada ciclo de conmutación.

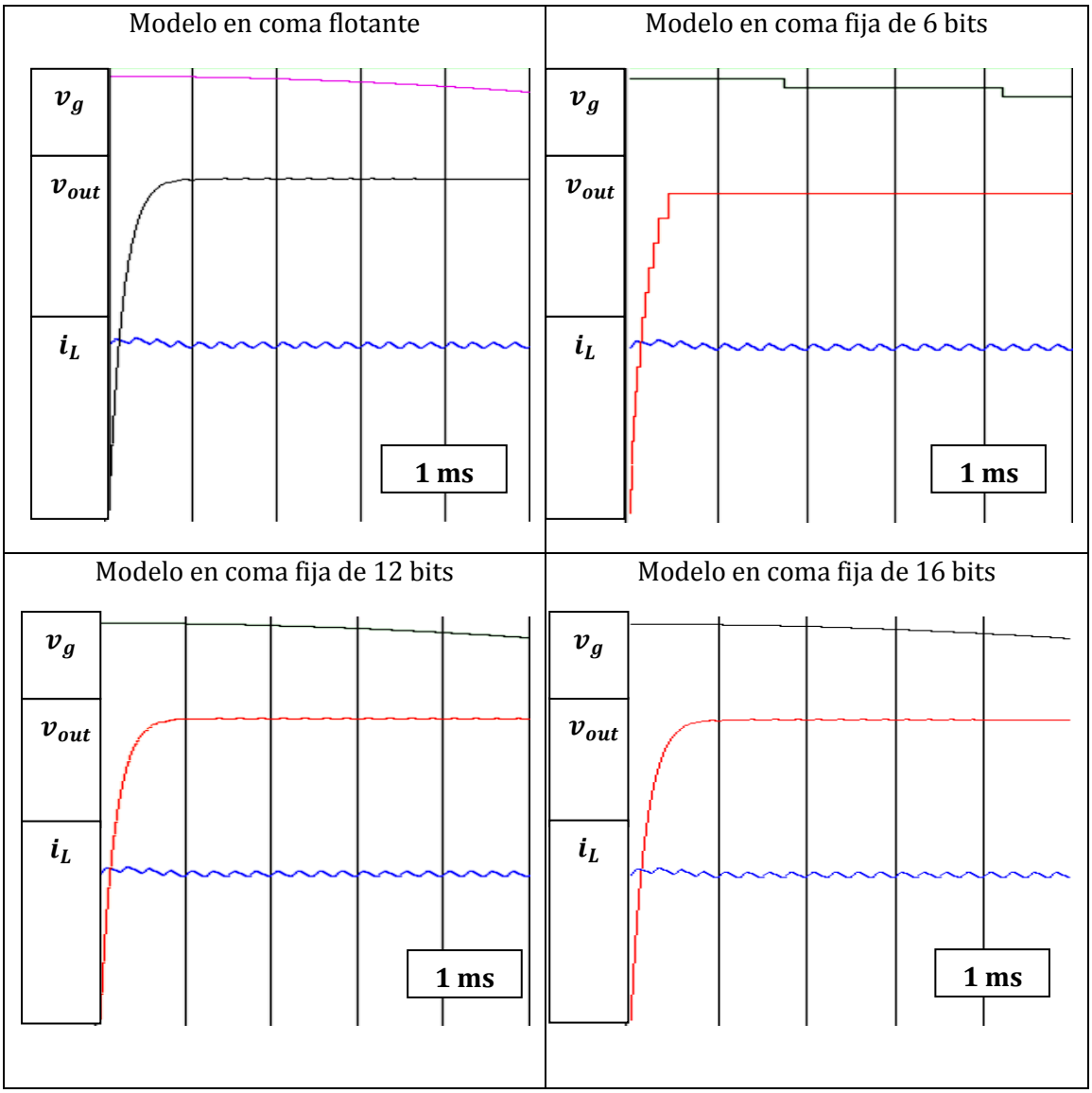

Figura 14. Zoom simulación del convertidor *Buck* en lazo abierto con 30% de ciclo de trabajo

A simple vista, fijándose en las figuras, los cuatro modelos simulados actúan de la misma manera, obteniendo dinámicas similares. Sin embargo, es imprescindible analizar resultados numéricos de los modelos. Para ello, las tablas 3 y 4 recogen muestras de las variables de estado en puntos característicos de la simulación, tales como puntos máximos y mínimos locales de las variables de estado.

Tiempo [ms]	0,0042	0,1417	0,218	8,0663
Real [V]	24,153	214,557	216,936	216,883
QX.Y 6 bits [V]	32	208	208	208
QX.Y 12 bits [V]	24,5	214,75	217	217
QX.Y 16 bits [V]	24,5625	214,688	216,906	216,813

Tabla 3. Valores de la tensión de salida en lazo abierto con 30% de ciclo de trabajo.

Tiempo [ms]	0,0042	0,0168	0,0505	0,0674	0,0969	0,118	0,1475	0,1643	0,1894	0,2149
Real [V]	1,0813	3,8936	1,8718	4,4106	1,3067	3,4836	0,1292	2,6299	0	2,4899
QX.Y 6 bits [V]	1,2353	3,7822	1,7089	4,3408	1,2177	3,5352	0,2441	2,5625	0	2,4248
QX.Y 12 bits [V]	1,1572	3,8466	1,7764	4,3457	1,4307	3,459	0,3086	2,5244	0	2,5742
QX.Y 16 bits[V]	1,1572	3,8466	1,7783	4,3467	1,4336	3,4609	0,3125	2,5303	0	2,5752

Tabla 4. Valores de la corriente de entrada en lazo abierto con 30% de ciclo de trabajo.

Como podemos observar en la tabla 3, el modelo del convertidor reductor implementado teniendo entradas de 6 bits, no llega a tener la precisión necesaria para funcionar como el modelo del convertidor *real*. Lo vemos, por ejemplo, en que el máximo valor que alcanza la tensión de salida para este caso es de 208 V, mientras que en el modelo *real* es 216,9 V. Esto es debido a que la resolución que se obtiene al utilizar solamente 6 bits de entrada, no es suficiente, ya que con tan pocos bits, se eliminan decimales, obteniendo un valor que no es el adecuado para el fin de este trabajo. De hecho, se puede observar que una vez que la tensión de salida se ha estabilizado, el modelo de 6 bits mantiene la tensión de salida a 208 V, no representando el rizado debido a la conmutación del convertidor. Por tanto, es evidente que este modelo carece de utilidad.

En cambio, en el mismo ejemplo, en el caso de entradas de 12 y 16 bits la tensión que se obtiene es de 216 V y 216,906 V, respectivamente. Es un valor realmente similar al *real*, por lo que estos dos modelos, son adecuados para sustituir al modelo en coma flotante, ya que como se comentó anteriormente, la coma fija aporta ventajas al modelado. Se ha calculado el error medio de tensión para los tres modelos en coma fija para verlo de forma más clara y se han obtenido como resultados 7,1875 V para 6 bits, 0,812 V para 12 bits y 0,77625 V para el de 16 bits.

Por último, en el caso de la corriente de entrada, plasmado en la tabla 4, los valores recogidos de los tres modelos en coma fija, son muy similares al modelo en coma flotante. Para verlo de manera más clara, se ha realizado el cálculo del error medio para la corriente de entrada de cada modelo en coma fija con respecto al modelo de tipo real. Para el QX.Y de 6 bits, el error medio es de 0,089 A, para el de 12 bits es 0,080 y para el de 16 bits es 0,080. Nuevamente se ven los resultados esperables al obtener un error decreciente según se aumentan el ancho de las entradas. En cuanto a la corriente de entrada, los tres modelos se comportan de forma correcta, aunque el modelo de 6 bits queda descartado por el cálculo de la tensión de salida, como se vio anteriormente.

3.1.2 CICLO DE TRABAJO FIJO 50%

También se han realizado pruebas del convertidor con otro valor de ciclo de trabajo, siendo en este caso el 50%, a diferencia del apartado 3.1.1 en el que era del 30%. Al variar el ciclo, se está variando el tiempo en el que el interruptor se encuentra abierto o cerrado, teniendo que obtener corrientes y tensiones más altas en este caso. De hecho, para este ciclo de trabajo no se entrará en modo de conducción discontinua, sino que el convertidor trabajará en modo de conducción continua en todo momento.

La figura 15 muestra gráficamente las variables de estado para los cuatro modelos, al igual que se hizo en el anterior apartado. Así se puede observar la dinámica de los modelos de forma gráfica. Es destacable el comportamiento de la tensión de salida en el modelo de 6 bits. Observando la figura 15 parece que la tensión de salida está representa con un trazo grueso. Sin embargo, lo que ocurre es que la resolución de dicha señal es tan pequeña que realmente la señal está oscilando continuamente entre dos valores consecutivos pero que realmente representan una diferencia grande de voltios, pareciendo que hay un trazo grueso en la representación de la señal.

Al igual que en el caso del ciclo de trabajo del 30%, se han tomado muestras de las variables de estado en diferentes puntos para comprobar numéricamente la precisión de los modelos. Éstas se recogen en las tablas 5 y 6.

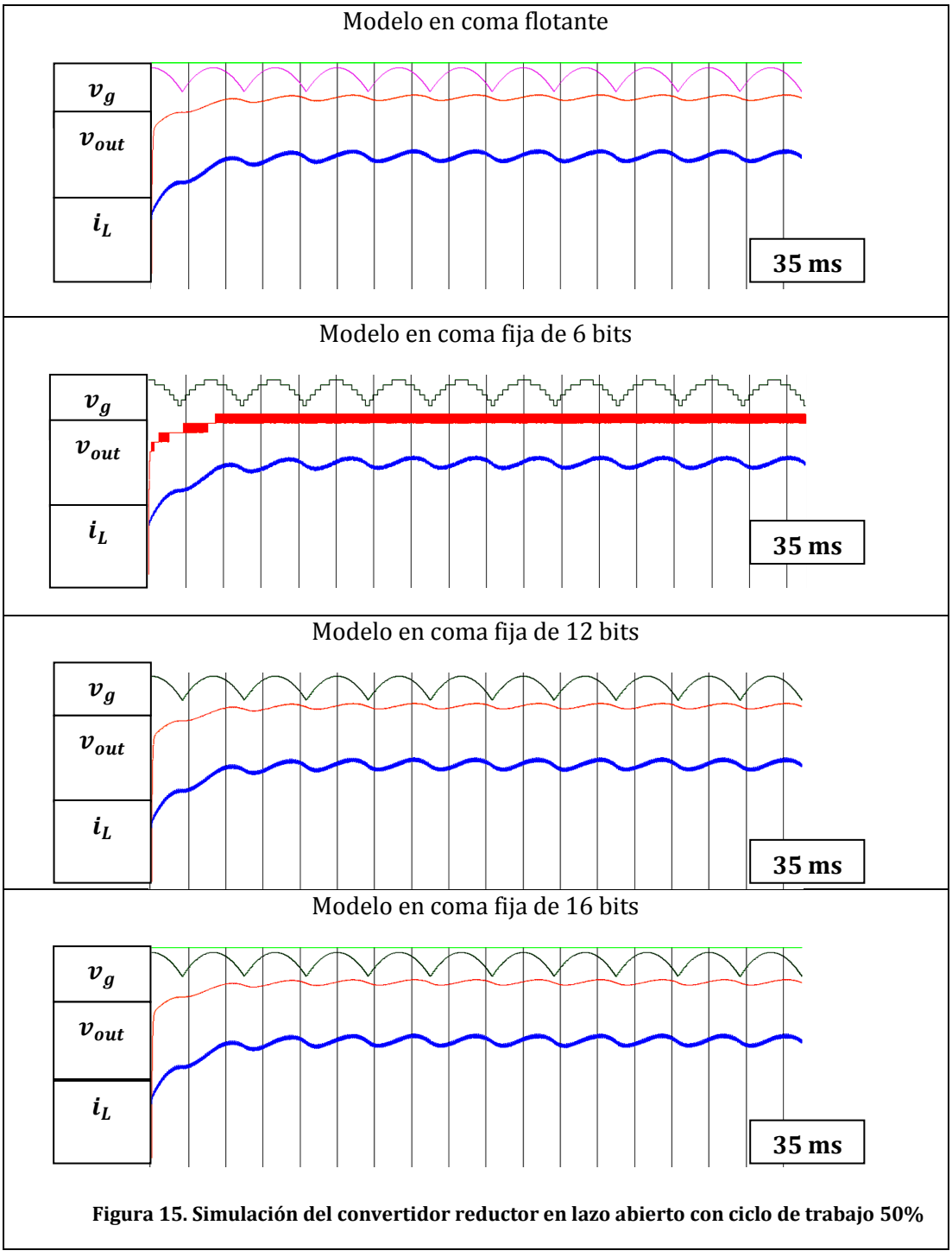

Figura 15. Simulación del convertidor reductor en lazo abierto con ciclo de trabajo 50%

Tiempo [ms]	0,0084	0,1758	1,6502	4,1269	5,6278	7,3289	8,8948
Real [V]	43,626	223,978	247,217	267,041	262,273	271,595	264,347
QX.Y 6 bits [V]	48	208	240	272	256	272	256
QX.Y 12 bits [V]	46,5	224,5	247	267	262,25	271,75	264,25
QX.Y 16 bits [V]	46,5	224,484	247,094	267,016	262,25	271,75	264,235

Tabla 5. Valores de la tensión de salida en lazo abierto con ciclo de trabajo del 50%

Tiempo [ms]	0,008	1,5219	4,3736	5,4309	7,5777	9,0216
Real [V]	2,1199	32,5774	52,795	47,367	57,0151	49,6514
QX.Y 6 bits [V]	1,9883	34,5439	55,917	50,0332	60,3203	52,9219
QX.Y 12 bits [V]	2,1611	32,3291	52,6504	46,6602	56,9502	49,5684
QX.Y 16 bits [V]	2,2705	32,7139	52,8008	47,1953	57,1836	49,7598

Tabla 6. Valores de la corriente d entrada en lazo abierto con ciclo de trabajo del 50%

Como podemos observar en las tablas 5 y 6, el modelo del convertidor reductor implementado con entradas de 6 bits, no llega a tener la resolución necesaria para funcionar como el modelo del convertidor real, ni en el caso de la tensión de salida ni en el de la corriente de entrada. El error medio de corriente ha sido calculado para esta prueba, igualmente que para el apartado 3.1.1, dando como valores 2,410 A para el modelo QX.Y de 6 bits, 0,215 A para 12 bits y en el de 16 bits, 0,124 A. Por otro lado, el error medio de tensión da valores de 6,793 V para el caso de 6 bits, 0,561 V para 12 bits y para 16 bits es 0,545 V. Por lo tanto, con estos resultados, queda reflejado nuevamente que el ancho de las señales de 6 bits, no es suficiente para medir las variables de estado. En cambio, los otros dos modelos en QX.Y tienen un error bajo, que permite su utilización. Nuevamente el de 16 bits es menor que el de 12, lo que es el resultado esperable.

3. 2 LAZO CERRADO

En el caso de los sistemas de control en lazo cerrado, al contrario que en lazo abierto, se produce realimentación, midiendo la salida de la planta. Esta medida se utiliza para recalcular la actuación y así llegar al resultad deseado. La ventaja del lazo cerrado, es que es más preciso y estable a perturbaciones y variaciones internas que el lazo abierto, pero es más complejo.

Para realizar las pruebas de este apartado, se ha integrado un regulador encargado de controlar la corriente media del convertidor reductor. Para ello se declara una corriente de referencia que permitirá ser comparada con la medida real, y así obtener el error que el regulador tratará de anular cambiando el ciclo de trabajo del convertidor. En los siguientes apartados se mostrarán los cálculos de precisión de los modelos para tres consignas de corriente diferentes: 13 A, 30 A y 5 A.

3.2.1 CORRIENTE DE REFERENCIA DE 13 A

En este apartado, se va a realizar la medida de las variables de estado, corriente de entrada y tensión de salida, en un sistema en lazo cerrado con una corriente de referencia media de 13 A.

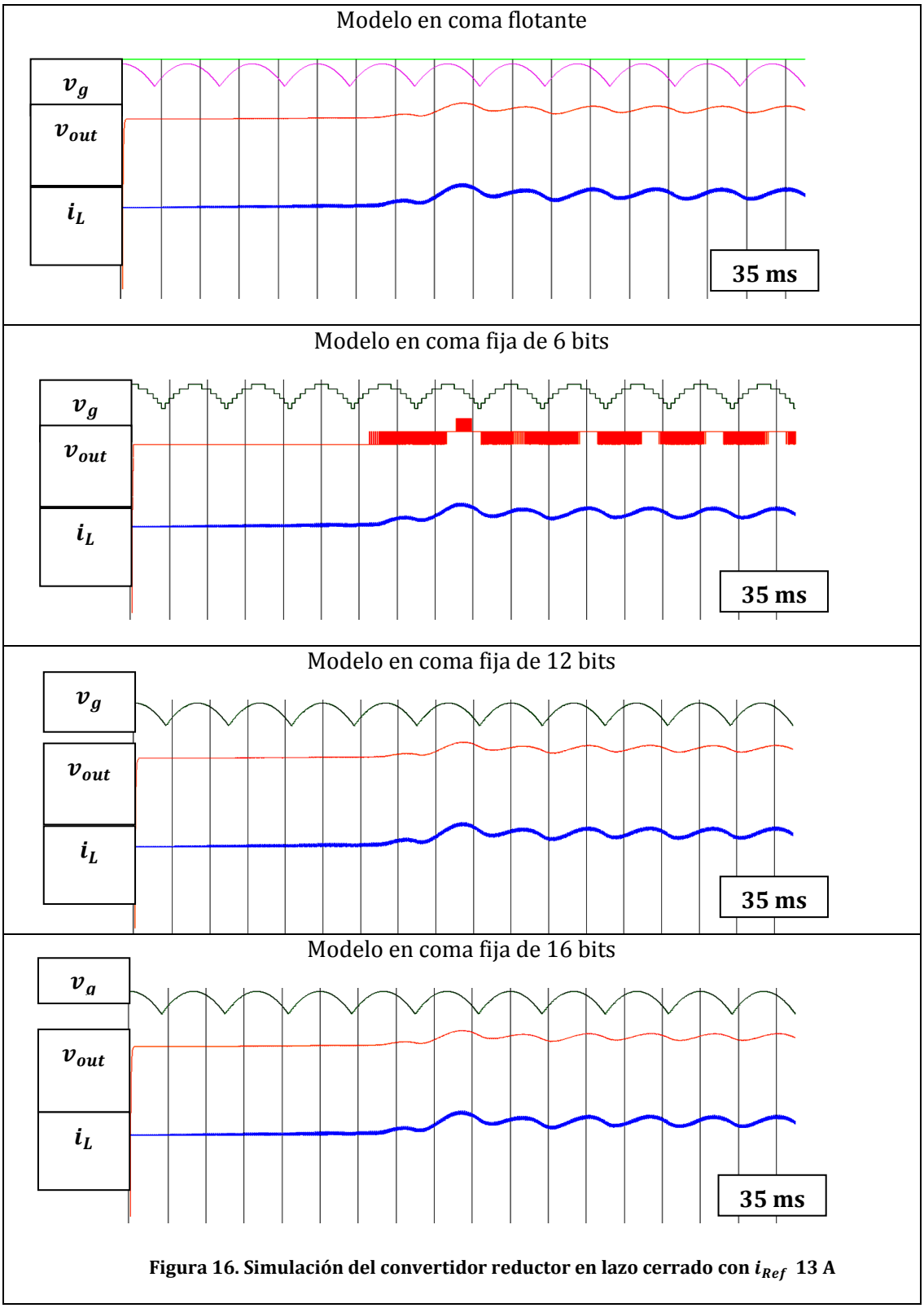

Figura 16. Simulación del convertidor reductor en lazo cerrado con i_{Ref} 13 A

La figura 16 muestra gráficamente la dinámica de los cuatro modelos en este experimento cuando, partiendo del reposo, reciben una consigna de 13 A. En este caso, al igual que en el apartado 3.1.2, hay que destacar el comportamiento de la tensión de salida en el modelo de 6 bits. Como puede verse, ésta no representa bien su rizado, como sí lo hacen los otros tres modelos. De hecho, puede observarse el mismo efecto del apartado

3.1.2, en el que la tensión de salida oscila entre dos o tres valores consecutivos pero que representan incrementos de varios voltios cada uno. Como ya se comentó, esto es debido a la escasa resolución del modelo con 6 bits.

Tiempo [ms]	0,0084	0,2432	13,2928	14,484	15,2685	17,4754	19,373	20,7729	22,4085	24,3171
Real [V]	43,403	215,795	218,306	222,088	219,696	235,697	226,462	230,873	222,361	231,288
QX.Y 6 bits [V]	48	208	208	224	208	224	224	224	208	224
QX.Y 12 bits [V]	46,25	216	218,25	222	219,75	235,75	226,5	231	222,5	231,5
QX.Y 16 bits [V]	46,256	215,797	218,344	222,063	219,5	235,531	226,359	230,813	222,359	231,453

Tabla 7. Valores de la tensión de salida en lazo cerrado (i_{Ref} 13 A)

Tiempo [ms]	0,0084	13,1249	14,4734	15,3246	17,4232	19,0751	20,6228	22,1735	24,1709	25,4261
Real [V]	0	3,10437	7,075	5,237	20,4491	11,347	15,9068	7,7836	16,8272	9,2876
QX.Y 6 bits [V]	0	3,3291	8,9717	6,4473	20,4043	10,3984	15,8047	8,2334	17,2822	9,5713
QX.Y 12 bits [V]	0	2,9824	7,1748	4,9951	20,9316	10,9102	16,2773	7,706	17,2705	9,0537
QX.Y 16 bits [V]	0	2,5712	6,77539	5,0937	20,4775	10,4609	16,0752	7,5322	17,1504	8,7988

Tabla 8. Valores de la corriente de entrada en lazo cerrado (i_{Ref} 13 A)

Para el caso que se presenta en este apartado, el error medio de la corriente obtenido para un ancho de 6 bits es 0,562 A, para 12 bits es 0,251 A y para 16 bits es 0,312 A. El error medio de tensión es 7,899 V para 6 bits, 0,382 V para 12 bits y, por último, 0,361 V para 16 bits. Se puede ver que los modelos en coma fija de 12 y 16 bits se comportan de manera parecida, alcanzando valores de error medio casi iguales. Fijándonos en la corriente, el modelo de 12 bits obtiene un mejor resultado, ya que por temas de redondeos ha salido favorecido, lo que haría que se eligiese éste como modelo sustituto. En cambio, en el cálculo del error medio de tensión el de 16 bits tiene un valor menor, que es lo que sería de esperar, así que en los dos apartados siguientes se comprobará el error medio de las pruebas y según los resultados se llegará a la mejor solución.

3.2.2 CORRIENTE DE REFERENCIA DE 30 A

En este apartado, se realizará una simulación del sistema en lazo cerrado aplicando una corriente de referencia de 30 A.

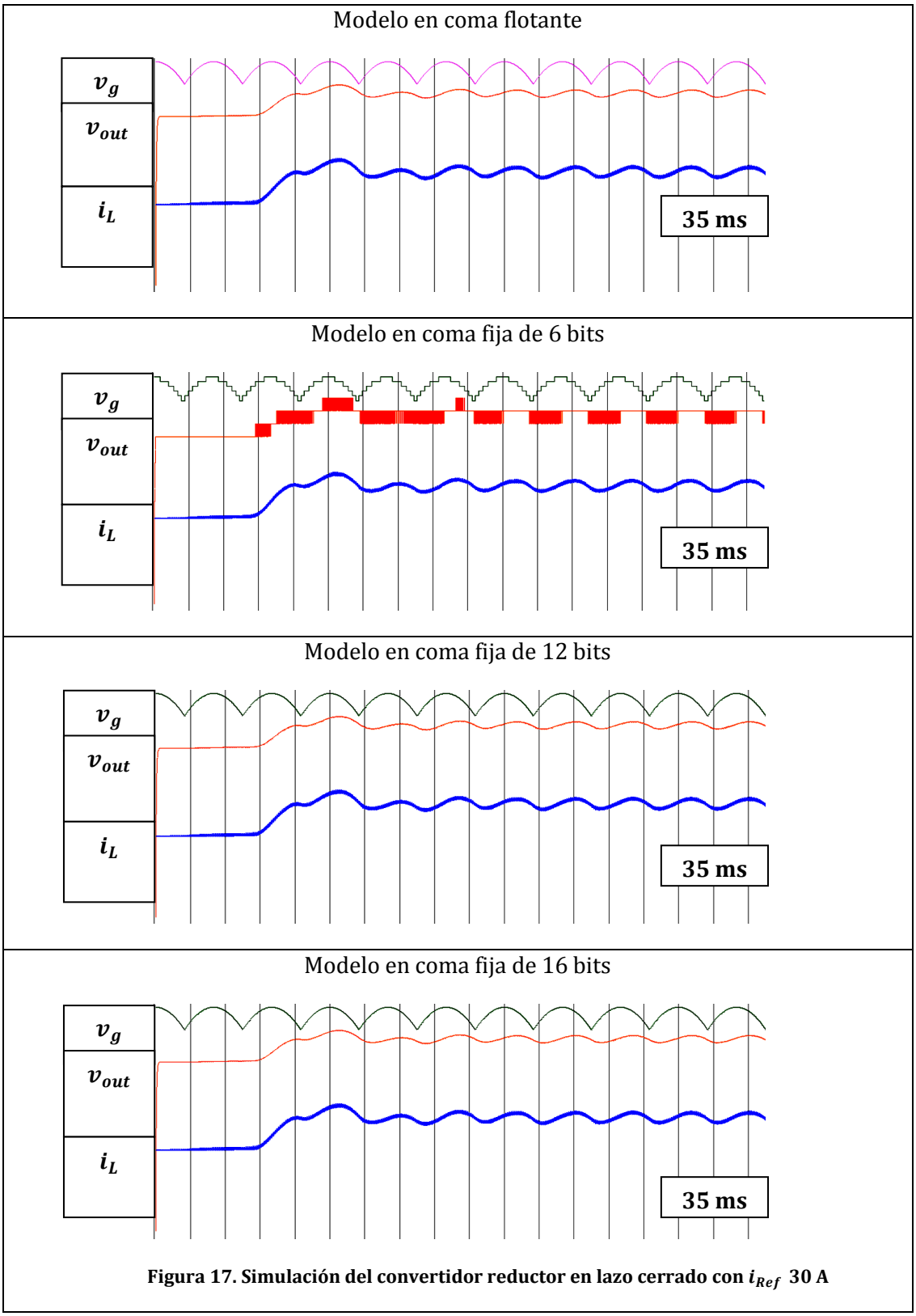

Figura 17. Simulación del convertidor reductor en lazo cerrado con i_{Ref} 30 A

Al realizar la simulación con una corriente de 30 A, se obtienen los resultados mostrados en la figura 17. En esta simulación, se vuelve a observar el efecto de la escasa resolución de bits en el caso del modelo del convertidor en coma fija de 6 bits, apareciendo un trazo grueso en el voltaje de salida de la segunda fila.

Tiempo [ms]	0,0084	0,2429	5,8999	7,9445	10,2747	12,5265	14,2673	15,7498	17,4279
Real [V]	43,4028	215,796	217,976	243,788	254,778	240,345	245,936	239,446	249,814
QX.Y 6 bits [V]	48	208	208	224	240	224	240	224	240
QX.Y 12 bits [V]	46,25	216	218	243,75	255	240,5	246	239,25	250
QX.Y 16 bits [V]	46,25	215,797	218,672	243,953	255	240,391	245,781	239,375	249,797

Tabla 9. Valores de la tensión de salida en lazo cerrado (i_{Ref} 30 A)

Tiempo [ms]	0,0084	5,8236	8,1245	10,5249	12,32	14,1236	15,4299	17,524	19,0325	20,7193
Real [V]	0	3,3295	29,6141	40,5214	25,576	31,7891	23,8015	34,8918	26,142	33,8928
QX. Y 6 bits [V]	0	3,7266	30,5703	40,4307	25,3662	31,6182	24,3086	35,3086	26,1006	34,4395
QX.Y 12 bits [V]	0	3,3369	29,6973	40,6758	25,5186	31,6621	23,6484	35,0625	25,2725	33,8193
QX.Y 16 bits [V]	0	3,3418	29,7246	40,8818	25,2754	31,7109	23,8594	35,0664	26,3057	34,118

Tabla 10. Valores de la corriente de entrada en lazo cerrado (i_{Ref} 30 A)

Calculando el error medio de la corriente de la tabla 10, tomando como referencia el modelo de tipo real, se han obtenido los valores de 0,310 A, 0,180 A y 0,140 A, para los QX.Y de 6, 12 y 16 bits, respectivamente. Ahora se procede a mostrar los resultados al calcular el error medio de tensión de la tabla 9, obteniendo 11,608 V para el modelo de 6 bits, 0,437 V para 12 bits y para 16 bits es 0,469 V. La diferencia entre los tres modelos para el caso de la corriente no es muy grande, pero como ha pasado en los apartados anteriores, el modelo de 6 bits sigue siendo descartado, ya que los valores calculados en el error medio de tensión no son buenos. Fijándose en los resultados de los modelos de 12 y 16 bits de la corriente, se ve que vuelve a ser menor el segundo, en cambio, mirando la tensión es menor el modelo de 12 bits. Luego se esperará a los resultados del siguiente apartado para llegar a una conclusión final de qué modelo es el más adecuado para ser sustituido por el real.

3.2.3 CORRIENTE DE REFERENCIA DE 5 A

Por último, se realizará una prueba en la que se simularán los modelos del convertidor *Buck* tomando como corriente de referencia de 5 A, para el cálculo de la realimentación.

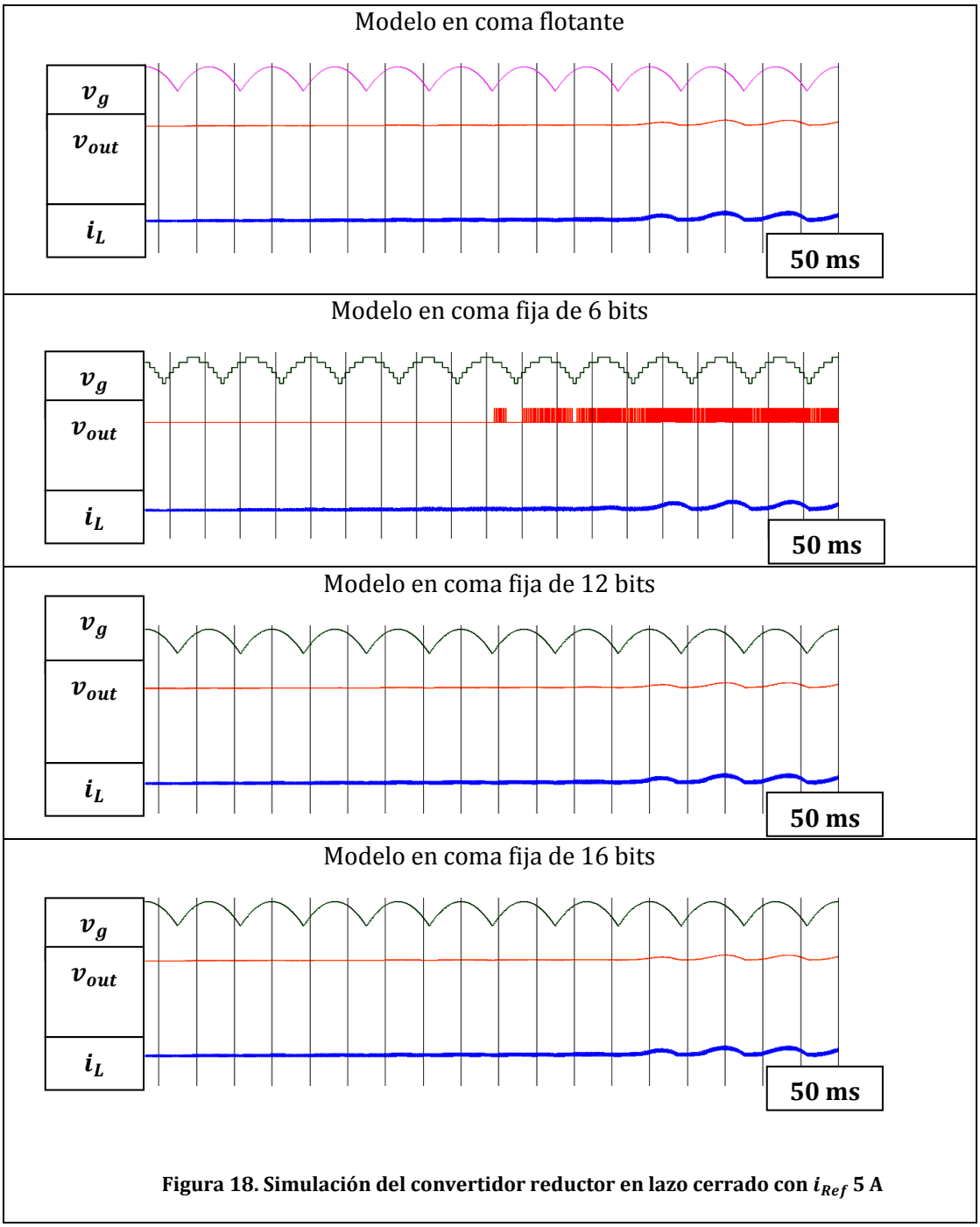

Figura 18. Simulación del convertidor reductor en lazo cerrado con i_{Ref} 5 A

En la figura 18, se muestran los resultados de las simulaciones realizadas. Se ha simulado un tiempo mayor que en los otros dos casos (apartados 3.2.1 y 3.2.2), pues la corriente de entrada y la tensión de salida se estabilizan a los 45 ms, aproximadamente. Como se puede ver, según las pruebas realizadas en el apartado 3.2, cuando se disminuye el valor de la corriente de referencia, el tiempo que tarda en estabilizarse las señales es mayor, y los valores máximos de éstas son menores. El trazo grueso que aparecía en las anteriores simulaciones, sigue apareciendo en ésta, ya que el rizado no llega a ser representado de forma correcta debido al truncado de la señal por el bajo número de bits.

Como en todos los casos anteriores, tanto en lazo abierto como en lazo cerrado, se han tomado muestras de las variables de estado en puntos característicos para calcular la precisión de los modelos numéricamente.

Tiempo [ms]	0,0084	0,2433	39,9223	40,5219	42,0985	44,0708
Real [V]	43,4028	215,794	219,082	220,437	217,425	222,692
QX.Y 6 bits [V]	48	208	208	224	208	224
QX.Y 12 bits [V]	46,25	216	219,25	220,75	217,25	223
QX.Y 16 bits [V]	46,25	215,797	219,219	220,656	217,281	222,859

Tabla 11. Valores de la tensión de salida en lazo cerrado (i_{Ref} 5 A)

Tiempo [ms]	0,0084	39,57	40,6235	41,6685	44,0697
Real [V]	0	3,7059	5,9634	2,3224	8,4025
QX.Y 6 bits [V]	0	4,1191	7,5547	2,25	8,5518
QX.Y 12 bits [V]	0	3,73926	5,8418	2,51953	8,35156
QX.Y 16 bits [V]	0	3,64844	6,02832	2,51172	8,39746

Tabla 12. Valores de la corriente de entrada en lazo cerrado (i_{Ref} 5 A)

Se han determinado los valores de error medio para los resultados de la tabla 12, obteniendo 0,445 A, 0,081 A y 0,063 A, para los modelos en coma fija de 6, 12 y 16 bits, respectivamente. Por otro lado se ha calculado el error medio de tensión para los modelos anteriores y se ha obtenido 6,295 V para el de 6 bits, 0,669 V para el de 12 y 0,586 V para el de 16.

Como se comentó en el apartado 3.1.2 y comparando los resultados de los errores medios de las 5 pruebas realizadas, los modelos con un ancho de 12 y 16 bits podrían ser utilizados para sustituir al modelo de tipo *real*, ya que los valores de los errores son pequeños. En una prueba es mejor el de 12 bits, en otras tres el de 16 bits y en una el error es prácticamente el mismo. Según esto, ya que los errores medio calculados no son muy diferentes entre el modelo de 12 y 16 bits, se elegirá el modelo en coma fija de ancho 12 bits, ya que el número de bits es menor.

3.3 TIEMPOS DE SIMULACIÓN Y EMULACIÓN

En los anteriores apartados se ha comprobado si los tres modelos en coma fija tienen suficiente resolución para actuar como modelos fiables de un convertidor reductor. Como se vio en las pruebas, el modelo con entradas de 6 bits carece de suficiente precisión para ser fiable. Sin embargo, los modelos de 12 y 16 bits actúan de forma similar con

resultados numéricos fiables, aunque matemáticamente el de 16 bits siempre será más fiable que el de 12 bits.

Una vez que se sabe que los modelos de 12 y 16 bits pueden usarse para modelar el convertidor reductor, es necesario saber en cuánto tiempo generarán resultados, es decir, qué velocidad de simulación y emulación obtienen los modelos en coma fija. Por tanto, en este apartado se realizará una comparativa de los cuatro modelos del convertidor reductor propuestos en lo referente al tiempo de simulación. Además, se procederá a comparar los tres modelos en coma fija según el tiempo de emulación, ya que el modelo real no puede ser emulado, y los recursos ocupados en la FPGA, tanto del modelo *reductor* aislado, como de toda la aplicación en conjunto.

Los modelos del convertidor en coma flotante y en coma fija han sido implementados en una FPGA Xilinx XC3S1000-4FT256 y las simulaciones se han realizado con la herramienta ModelSim 6.5b de Mentor Graphics. El ordenador utilizado, para la medida de los tiempos, ha sido un Intel Core i7 – 4770 a 3,40 GHz con una memoria RAM de 8.0 GB.

	100 ms	**200 ms**	**Aceleración media**
Real	57''89	1'53''96	
QX.Y 6 bits	5'43''53	11'33''34	0,166x
QX.Y 12 bits	4'51''72	9'48''30	0,196x
QX.Y 16 bits	5'18''01	10'41''40	0,180x

Tabla 13. Tiempo de simulación de los cuatro modelos del convertidor reductor

En la tabla 13, se muestran los resultados del tiempo de simulación de los cuatro modelos del convertidor reductor, habiendo simulado 100 ms y 200 ms. Como se puede observar, a la hora de simular, el modelo del convertidor en coma flotante, tipo *real*, se efectúa de manera más rápida que cualquier caso del modelado en coma fija. Se ha añadido una columna a la tabla en la que se indica la aceleración media de los modelos. Para el caso de la simulación, como se ha dicho, los resultados son menores que la unidad, lo que indica que no ha aumentado la velocidad comparando con el modelo de referencia, sino que ha disminuido. Por otro lado, se puede observar que, centrándose sólo en los resultados de QX.Y, el caso de coma fija de 12 bits, es el más rápido de los tres, con una aceleración de 0.196x, frente al modelo *real*, seguido del de 16 bits, siendo el más lento el de 6 bits, aunque todos están alrededor de los 5 minutos. Puede verse, además, que no existe una relación directa y clara entre el número de bits y el tiempo de simulación. Lo que queda claro es que no tiene sentido simular los modelos en coma fija. Esto no supone ninguna contradicción con la propuesta de este Trabajo Fin de Grado, dado que los modelos en coma fija realmente tienen utilidad cuando son emulados.

	Emulación		Aceleración media
	100 ms	200 ms	
HIL QX.Y 6 bits	347,61 ms	695,22 ms	165,228x
HIL QX.Y 12 bits	389,16 ms	778,32 ms	147,587x
HIL QX.Y 16 bits	389,89 ms	779,78 ms	147,311x

Tabla 14. Tiempo de emulación de los modelos del convertidor reductor para 100 y 200 ms

Por ello, en la tabla 14 se presenta el tiempo de emulación de los tres modelos del convertidor reductor en coma fija. Como se puede ver en la tabla, el número de bits no afecta mucho al tiempo de emulación. Esto es debido a que en lo que más tiempo se emplea es en realizar las multiplicaciones, en las que se han utilizado multiplicadores embebidos de 18x18 bits para los tres casos, por lo que el número de bits de entrada pasa a un segundo lugar. Es decir, aunque cada modelo tiene diferentes anchos en las variables de estado, ya se tuvo en cuenta lo costosa que serían las multiplicaciones, por lo que en todos los casos se redujeron a 18 bits dichas variables para la multiplicación. Como ya se comentó a la hora de presentar el modelo en coma fija, esta reducción de bits no tiene grandes efectos en la precisión de los modelos, ya que la multiplicación se usa para conocer un valor instantáneo de tensión o corriente, y no para la propia integración del modelo.

En la tercera columna de la tabla 14, se ha detallado la aceleración de emular el sistema respecto a simularlo con el modelo *real*. Además, se ha añadido una comparación de la aceleración del sistema en coma fija en relación con el modelo de tipo *real*, como se realizó en la tabla 13. Como puede observarse, la gran diferencia se encuentra en la reducción de tiempo en la emulación, en comparación con la simulación, alcanzando valores de aceleración alrededor de 147,5 veces mayor que el modelo *real* para los casos de coma fija con 12 y 16 bits de entrada, así como de 165,228 para el caso de 6 bits.

Como se comentó con anterioridad, las diferencias de tiempo entre los modelos QX.Y son muy bajas, pero basándonos únicamente en el tiempo de simulación y emulación, sería recomendable quedarse con el modelo del convertidor reductor en coma fija con entradas de 6 bits, pues es el más rápido y el que menos bits utiliza. Sin embargo, como se ha demostrado en las pruebas en lazo abierto y lazo cerrado, realizadas en los apartados 3.1 y 3.2, la precisión del sistema en coma fija de 6 bits no es suficiente para modelar el convertidor reductor de forma correcta. Por tanto, se deberían utilizar los modelos de 12 bits o el de 16 bits, pues con éstos se consiguen valores de tensión y corriente muy similares al *real*, a la vez que se consiguen grandes aceleraciones respecto a la simulación *real*.

3.4 RESULTADOS DE IMPLEMENTACIÓN

Ya se ha comprobado la precisión y tiempos de simulación y emulación de los cuatro modelos propuestos. En el apartado 3.1 y 3.2 se comprobó que el modelo en coma fija con entradas de 6 bits no ofrecía suficiente precisión por lo que no es útil para el modelado del convertidor. Por otra parte, en el apartado 3.3 se comprobaron los tiempos de simulación y emulación, mostrando que cualquier sistema emulado es hasta 166 veces más rápido que el uso del modelo *real*, y que el número de bits de las entradas no hace variar en gran medida la velocidad de emulación del sistema. Por tanto, tanto el sistema de 12 bits como el de 16 bits parecen razonables para su uso.

Por último se van mostrar los resultados de área de la implementación de los distintos modelos en coma fija. Aunque a priori puede parecer una cuestión menor, es importante saber si es necesaria una FPGA grande para implementar la emulación. Para analizar bien el sistema de emulación, éste se ha dividido en dos subsistemas: el modelo del reductor, que es la propuesta de este TFG y el resto de componentes (regulador, modelo de ADCs, interconexiones, etc). Los resultados de implementación se recogen en la tabla 15, y se muestran por separado para ver realmente qué recursos usa el modelo y qué recursos hacen falta para la emulación del sistema completo. De esa forma se podrá saber qué aumento de recursos se produce cuando se aumenta el número de bits del modelo.

	LUTs 4 entradas	Flip Flops	Mult.18x18	BRAM
Buck-6 bits	419	41	2	0
Buck-12 bits	580	54	2	0
Buck-16 bits	692	62	2	0
HIL-6 bits	837	252	4	1
HIL-12 bits	1267	263	4	1
HIL-16 bits	1698	272	4	1

Tabla 15. Recursos ocupados en la FPGA Xilinx XC3S1000

Como era de esperar, los modelos del reductor usan dos multiplicadores, uno para deshacer la transformación de la tensión de salida y así poder realimentar el modelo, y otra para calcular la corriente de entrada para el regulador. Por otra parte, la cantidad de registros (*flip flops*) viene determinado por el ancho de las variables de estado del modelo (tensión de salida y corriente de entrada), pues son las señales que están registradas. En cuanto a las BRAM (Block RAM), son memorias alojadas en la FPGA que se han usado para guardar la tensión de entrada trifásica. Dado que la tensión de entrada es periódica, se han precalculado una serie de muestras y se han guardado en una BRAM. Por ello, en todos los sistemas HIL se usa una memoria. El uso de estos valores precalculados permite eliminar una gran cantidad de cálculos en tiempo real, que sería costosos en términos de área y posiblemente en frecuencia.

Comparando los recursos del modelo con los recursos del sistema HIL, se puede observar que el modelo ocupa menos del 50% de los recursos del diseño total. Por tanto, añadir un modelo de un convertidor reductor a un regulador para emularlo ocupa pocos recursos. Además es destacable que la emulación ocupa una porción pequeña de una FPGA de bajo coste como es la FPGA Xilinx XC3S1000. Por tanto, para un modelo de un convertidor sencillo como es el reductor, el área ocupada es muy pequeña, no siendo éste un factor importante a la hora de diseñar el modelo.

3.5 CONCLUSIONES DE LOS RESULTADOS

En los apartados previos se han realizado pruebas de precisión, tiempos de simulación y área ocupada en emulación. Este apartado tiene como objetivo presentar un pequeño resumen de los resultados obtenidos.

En todos los casos se ha visto que un modelo de 6 bits de entrada no es viable para aplicaciones como la presentada en este TFG. Esto es debido a la baja resolución que ofrecen las entradas. Sin embargo, los modelos de 12 y 16 bits presentan resultados muy parecidos a los resultados del modelo *real*, al que hemos elegido como referencia, dado que tiene gran resolución. Por tanto, tanto los modelos de 12 como de 16 bits son válidos en cuanto a precisión, si bien es cierto que un mayor número de bits siempre será recomendable.

Otro factor clave es el tiempo de simulación y emulación de los modelos presentados, ya que la principal motivación de este TFG es la aceleración de las etapas de pruebas de un convertidor de potencia. Se ha visto que simular un modelo en coma fija es contraproducente, ya que tarda más que el modelo *real* y además es más difícil de diseñar. Por otra parte, los modelos en coma fija se pueden emular y ahí es donde se han cumplido los objetivos del TFG, ya que se han obtenido tiempos de emulación hasta 166 más cortos que el tiempo de simulación de un modelo *real*. Lo que sí se ha visto es que el número de bits de los modelos no afectan en gran medida a la velocidad de emulación siempre que se tengan consideraciones a la hora de implementarlos, como puede ser la optimización de los multiplicadores embebidos de la FPGA.

Por último, se ha comprobado el área que ocupan los modelos en la FPGA para ver si es viable realizar un modelo con un dispositivo de bajo coste. Los resultados de implementación han sido muy similares en todos los casos y se ha visto que el porcentaje de ocupación de la FPGA es muy reducido. Por tanto, este factor no es importante a la hora de realizar un modelo simplemente como el de un convertidor reductor.

Teniendo todos los factores en cuenta, se ha visto que tanto los modelos de 12 y 16 bits presentan buena precisión, alta velocidad y necesitan pocos recursos. Por tanto, cualquiera de los dos modelos sería adecuado para esta aplicación, si bien es cierto que siempre será más recomendable usar un modelo con mayor número de bits.

4. CONCLUSIONES

En este trabajo de fin de grado, se ha desarrollado un modelo de un convertidor reductor (*buck*). Este modelo ha sido diseñado para simular la etapa de carga de una batería para vehículos. Esta simulación tiene aplicación industrial ya que una batería de vehículo debe someterse a unos ciclos de carga-descarga antes de ser utilizada, para conseguir que funcione de manera óptima.

Para poder probar un regulador digital en un sistema real, se ha visto que es imprescindible realizar primero una simulación de éste, para estar seguros de que no se va a producir ningún daño ni material ni personal. Estas simulaciones son complejas ya que hay que simular partes digitales (regulador) junto a partes analógicas (convertidor). En el estado del arte analizado se ha visto que estas simulaciones pueden ser ejecutadas siguiendo esquemas diferentes, como simular por una parte el regulador, y por otra parte la planta, uniéndolos mediante un lenguaje de programación tipo C, lo que lleva trabajo adicional. También se ha visto la posibilidad de usar un simulador mixto, pero son caros y las simulaciones son muy lentas. En este trabajo se eligió modelar la planta al lenguaje VHDL, para así poder simular todo el sistema en conjunto de forma digital. Además, al codificar un modelo en VHDL existe la posibilidad de emular el sistema, es decir, ejecutar las pruebas con un dispositivo digital programable, como puede ser una FPGA en vez de realizarlo en *software*.

Para realizar el modelo se han desarrollado las ecuaciones en diferencias del circuito del convertidor reductor, y han sido utilizadas para realizar cuatro modelos del convertidor, uno en coma flotante (*real*) y tres en coma fija (*sfixed*), utilizando distinto número de bits en las entradas en cada uno de ellos. El número de bits en los modelos QX.Y fueron 6, 12 y 16 bits. Estos cuatro modelos han sido implementados en el programa Xilinx ISE y simulados con ModelSim para realizar las pruebas necesarias para conseguir el modelo óptimo para la etapa de carga simultánea de las 16 baterías de vehículos.

Como se ha visto, el modelo *real* es sencillo y rápido en la simulación, pero tiene el inconveniente de que no puede ser emulado. En cambio, los modelos en coma fija, permiten la simulación y emulación. El problema de éstos es que requieren de un mayor esfuerzo en el diseño, pues es necesario tener en cuenta el tamaño de las variables y su resolución. De hecho, se ha visto lo importante que es la resolución de las variables en un modelo, ya que los modelos integran pequeños valores para ofrecer una simulación precisa. Sin embargo, este trabajo adicional que requiere el modelado en coma fija se ve compensado en la gran aceleración de las pruebas.

Por último, en los resultados experimentales ha quedado demostrado que la emulación acelera hasta 166 veces la simulación *real*. También se ha demostrado la importancia de la resolución en los modelos, dado que se ha observado que el modelo 6 bits no es suficientemente preciso para la aplicación propuesta, mientras que los modelos de 12 y 16 bits consiguen precisión parecida, al igual que tiempos de emulación y uso de recursos *hardware* parecidos.

BIBLIOGRAFÍA

[1] J. Méndez Serrano, "Corrector de factor de potencia reductor-elevador monofásico", Tesis Profesional, 2004.

[2] Seminario Anual de Automática, Electrónica Industrial e Instrumentación, "Modelos de convertidores electrónicos de potencia para simulación por ordenador", Pamplona, 1998.

[3] A. Castro Martín, "Aplicación del control digital basado en hardware específico para convertidores de potencia conmutados", Tesis Doctoral, Universidad Politécnica de Madrid, 2003.

[4] SystemVision, "www.mentor.com", 2014.

[5] P. Zumel, M. García-Valderas, A. Lázaro, C. López-Ongil and A. Barrado, "Co-simulation Psim-Modelsim oriented to digitally controlled switching power converters" in *Control and Modeling for Power Electronics (COMPEL), 2010 IEEE 12th Workshop on*, pp. 1-7, jun. 2010.

[6] A. Sánchez González, "Aportaciones mediante implementación basada en sistemas embebidos al control digital de convertidores conmutados", Tesis Doctoral, Universidad Autónoma de Madrid, 2013.

ANEXO I. GLOSARIO

AC	*Alternating Current*
ADC	*Analog to Digital Converter*
BRAM	*Block RAM*
CCM	*Continuous Current Mode*
DC	*Direct Current*
DCM	*Discontinuous Current Mode*
FPGA	*Field Programmable Gate Array*
HDL	*Hardware Description Language*
HIL	*Hardware In the Loop*
LUT	*Look Up Table*
PWM	*Pulse Width Modulation*
RAM	*Read Only Memory*
VHDL	*VHSIC Hardware Description Language*

ANEXO II. LISTA DE CÓDIGOS

En este anexo se procede a presentar el código implementado para la realización del trabajo de fin de grado. Los ficheros que se encontrarán son los siguientes:

- Modelos del convertidor reductor
- Modelos del ADC, el cual modela la interfaz de ADC que espera el regulador y modela los retrasos de un ADC.
- Regulador, que es único para todas las simulaciones.
- Fichero Top, que es el encargado de unir los modelos del reductor y ADC con el regulador.
- Bancos de pruebas para la simulación del sistema.

II.1 MODELOS DEL CONVERTIDOR REDUCTOR

II.1.1 MODELO REAL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;

entity BuckConverter is
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Mosfet : in  STD_LOGIC; --SW
           Vg : in  real;
           Ir : in  real;
           Vout : out  real;
           Iin : out  real);
end BuckConverter;

architecture Behavioral of BuckConverter is

 constant dt : real := 10.0e-9;
 constant L : real := 0.002;--2mH
 constant C : real := 0.000035;
 constant il_inicial : real := 0.0;
 constant vout_inicial : real := 0.0;

 signal v_out, vout_aux : real;
 signal il, il_aux : real;

begin

 --Cálculo de las señales iL y vout
 process(MOSFET,Vg,v_out,il,Ir)
 begin
```

```
        --Interruptor cerrado
        if Mosfet = '1' then
              il_aux <= (Vg-v_out)*(dt/L);
              vout_aux <= (il-Ir)*(dt/C);
        --Interruptor abierto
        else
              if il <= 0.0 then
              --Circuito en DCM (Discontinuous Conduction Mode)
                    il_aux <= 0.0;
                    vout_aux <= -(Ir*(dt/C));

              else  --Circuito en CCM (Continuous Conduction Mode)
                    il_aux <= -(v_out*(dt/L));
                    vout_aux <= (il-Ir)*(dt/C);

              end if;
        end if;
  end process;

  process(Clk, Reset)
  begin
        if Reset = '1' then
              il <= il_inicial;
              v_out <= vout_inicial;

        elsif rising_edge(Clk) then
              il <= il + il_aux;
              v_out <= v_out + vout_aux;

        end if;
  end process;

  Iin <= il;
  Vout <= v_out;

end Behavioral;
```

II.1.2 MODELO EN COMA FIJA (QX.Y 6 BITS)

```
library IEEE, ieee_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;

use ieee_proposed.fixed_float_types.all;
use ieee_proposed.fixed_pkg.all;

entity BuckConverter is
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Mosfet : in  STD_LOGIC; --SW
           Vg : in  STD_LOGIC_VECTOR (6 downto 0);--Q10.-4
           Ir : in  STD_LOGIC_VECTOR (6 downto 0);--Q26.-20
           Vout : out  STD_LOGIC_VECTOR (6 downto 0);--Q10.-4
           Iin : out  STD_LOGIC_VECTOR (19 downto 0));--Q9.10
```

```
end BuckConverter;


architecture Behavioral of BuckConverter is

  constant dt : real := 10.0e-9;
  constant L : real := 0.002;--2mH
  constant C : real := 0.000035;

  constant dtL : sfixed(-13 downto -30) := to_sfixed(dt/L, -13, -30);--valor para ser multiplicado a iL* para obtener iL
  constant VOUTSCALE : sfixed(-23 downto -40) := to_sfixed((dt*dt)/(C*L),-23,-40);--valor para ser multiplicado a Vout* para obtener Vout

  constant il_inicial : sfixed(9 downto -10) := to_sfixed(0.0, 9, -10);--valor de iL* en caso de Reset=1
  constant vout_inicial : sfixed(10 downto 4) := to_sfixed(0.0, 10, 4);--valor de Vout* en caso de Reset=1

  signal v_out : sfixed (10 downto 4);--señal auxiliar de Vout para trabajar en sfixed
  signal vout_as : sfixed (39 downto 20);--valor de Vout* nada más salir del sumador, sin estar ajustado a Q38.-20
  signal Vg_aux : sfixed (10 downto 4);--señal auxiliar de Vg para trabajar en sfixed

  signal il : sfixed (9 downto -10); --señal auxiliar de Iin para tranajar en sfixed
  signal il_as : sfixed (26 downto 4);--valor de iL* nada más salir del sumador, sin estar ajustado a Q25.-4
  signal Ir_aux : sfixed (26 downto 20);--señal auxiliar de Ir para trabajar en sfixed

  signal il_aux : sfixed (10 downto 4);--ajustamos Vg-Vout a Q10.-4
  signal il_ajustar : sfixed (25 downto 4);--ajustamos el valor de iL* para entrar al FF a Q25.-4
  signal ilMult_ajustar : sfixed (25 downto 8);--ajustamos iL* a Q25.-8 para que tenga 18 bits y poder pasarlo al multiplicador
  signal ilas_ajustar : sfixed (26 downto 20);--ajustamos iL* a Q26.-20 para poder restarle iR

  signal vout_aux : sfixed (26 downto 20);--ajustamos iL*-iR a Q26.-20
  signal vout_ajustar : sfixed (38 downto 20);--ajustamos el valor de Vout* para entrar al FF a Q38.-20
  signal voutMult_ajustar : sfixed (38 downto 21);--ajustamos Vout* a Q38.-21 para que tenga 18 bits y poder pasarlo al multiplicador


begin

  Vg_aux <= to_sfixed (Vg,Vg_aux);--pasamos Vg de std_logic a sfixed para poder operar con el resto de señales
  Ir_aux <= to_sfixed (Ir,Ir_aux);--pasamos Ir de std_logic a sfixed para poder operar con el resto de señales
```

```
	process(Mosfet,Vg_aux,v_out,il,il_as,Ir_aux,il_ajustar,ilas_ajus
tar)
	begin
		if Mosfet = '1' then --interruptor cerrado(SW=1)
			il_aux <= resize(Vg_aux-v_out,il_aux);
			ilas_ajustar <= resize(il_ajustar,ilas_ajustar);
			vout_aux <= resize(ilas_ajustar-Ir_aux,vout_aux);

		else
			if il <= 0.0 then
				il_aux <= to_sfixed(0.0, il_aux);
				vout_aux <= resize(-Ir_aux,vout_aux);

			else
				il_aux <= resize(-v_out,il_aux);
				ilas_ajustar <=resize(il_ajustar,ilas_ajustar);

				vout_aux<=resize(ilas_ajustar-Ir_aux,vout_aux;

			end if;
		end if;
	end process;

	process(Clk, Reset)
	begin
		if Reset = '1' then
		  il_ajustar <= resize(il_inicial,il_ajustar);
		  vout_ajustar <= resize(vout_inicial,vout_ajustar);

		elsif rising_edge(Clk) then
		  il_ajustar <= resize(il_ajustar + il_aux,il_ajustar);
		  vout_ajustar<=resize(vout_ajustar+vout_aux,vout_ajustar);


		end if;
	end process;

	--sacar iL=(iL*)*(dt/L)
	ilMult_ajustar <= resize(il_ajustar,ilMult_ajustar);
	il <= resize(ilMult_ajustar*dtL, il);

	--sacar Vout=(Vout*)*(dt/L)*(dt/C)
	voutMult_ajustar <= resize(vout_ajustar,voutMult_ajustar);
	v_out <= resize(voutMult_ajustar*VOUTSCALE, v_out);


	Iin <= to_slv(il);
	Vout <= to_slv(v_out);


end Behavioral;
```

II.1.3 MODELO EN COMA FIJA (QX.Y 12 BITS)

```
library IEEE, ieee_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
```

```
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;

use ieee_proposed.fixed_float_types.all;
use ieee_proposed.fixed_pkg.all;

entity BuckConverter is
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Mosfet : in  STD_LOGIC; --SW
               Vg : in  STD_LOGIC_VECTOR (12 downto 0);--Q10.2
           Ir : in  STD_LOGIC_VECTOR (12 downto 0);--Q26.-14
           Vout : out  STD_LOGIC_VECTOR (12 downto 0);--Q10.2
           Iin : out  STD_LOGIC_VECTOR (19 downto 0));--Q9.10
end BuckConverter;

architecture Behavioral of BuckConverter is

 constant dt : real := 10.0e-9;
 constant L : real := 0.002;--2mH
 constant C : real := 0.000035;

 constant dtL : sfixed(-13 downto -30) := to_sfixed(dt/L, -13, -30);--valor para ser multiplicado a iL* para obtener iL
 constant VOUTSCALE : sfixed(-23 downto -40) := to_sfixed((dt*dt)/(C*L),-23,-40);--valor para ser multiplicado a Vout* para obtener Vout

 constant il_inicial : sfixed(9 downto -10) := to_sfixed(0.0, 9, -10);--valor de iL* en caso de Reset=1
 constant vout_inicial : sfixed(10 downto -2) := to_sfixed(0.0, 10, -2);--valor de Vout* en caso de Reset=1

 signal v_out : sfixed (10 downto -2);--señal auxiliar de Vout para trabajar en sfixed
 signal vout_as : sfixed (39 downto 14);--valor de Vout* nada más salir del sumador, sin estar ajustado a Q38.-14
 signal Vg_aux : sfixed (10 downto -2);--señal auxiliar de Vg para trabajar en sfixed

 signal il : sfixed (9 downto -10); --señal auxiliar de Iin para tranajar en sfixed
 signal il_as : sfixed (26 downto -2);--valor de iL* nada más salir del sumador, sin estar ajustado a Q25.2
 signal Ir_aux : sfixed (26 downto 14);--señal auxiliar de Ir para trabajar en sfixed

 --Señales para ajustar tamaños de las variables
 signal il_aux : sfixed (10 downto -2);--ajustamos Vg-Vout a Q10.2
 signal il_ajustar : sfixed (25 downto -2);--ajustamos el valor de iL* para entrar al FF a Q25.2
 signal ilMult_ajustar : sfixed (25 downto 8);--ajustamos iL* a Q25.-8 para que tenga 18 bits y poder pasarlo al multiplicador
 signal ilas_ajustar : sfixed (26 downto 14);--ajustamos iL* a Q26.-14 para poder restarle iR

 signal vout_aux : sfixed (26 downto 14);--ajustamos iL*-iR a Q26.-14
```

```
      signal vout_ajustar : sfixed (38 downto 14);--ajustamos el valor
de Vout* para entrar al FF a Q38.-14
      signal voutMult_ajustar : sfixed (38 downto 21);--ajustamos
Vout* a Q38.-21 para que tenga 18 bits y poder pasarlo al
multiplicador


     begin

      Vg_aux <= to_sfixed (Vg,Vg_aux);--pasamos Vg de std_logic a
sfixed para poder operar con el resto de señales
      Ir_aux <= to_sfixed (Ir,Ir_aux);--pasamos Ir de std_logic a
sfixed para poder operar con el resto de señales

      process(Mosfet,Vg_aux,v_out,il,il_as,Ir_aux,il_ajustar,ilas_ajus
tar)
      begin
            if Mosfet = '1' then --interruptor cerrado(SW=1)
                  il_aux <= resize(Vg_aux-v_out,il_aux);
                  ilas_ajustar <= resize(il_ajustar,ilas_ajustar);
                  vout_aux <= resize(ilas_ajustar-Ir_aux,vout_aux);

            else
                  if il <= 0.0 then
                        il_aux <= to_sfixed(0.0, il_aux);
                        vout_aux <= resize(-Ir_aux,vout_aux);

                  else
                        il_aux <= resize(-v_out,il_aux);
                        ilas_ajustar <=resize(il_ajustar,ilas_ajustar);
                        vout_aux<=resize(ilas_ajustar-Ir_aux,vout_aux);

                  end if;
            end if;
      end process;

      process(Clk, Reset)
      begin
            if Reset = '1' then
              il_ajustar <= resize(il_inicial,il_ajustar);
              vout_ajustar <= resize(vout_inicial,vout_ajustar);

            elsif rising_edge(Clk) then

              il_ajustar <= resize(il_ajustar + il_aux,il_ajustar);
              vout_ajustar<=resize(vout_ajustar+vout_aux,vout_ajustar);


            end if;
      end process;

      --sacar iL=(iL*)*(dt/L)
      ilMult_ajustar <= resize(il_ajustar,ilMult_ajustar);
      il <= resize(ilMult_ajustar*dtL, il);

      --sacar Vout=(Vout*)*(dt/L)*(dt/C)
      voutMult_ajustar <= resize(vout_ajustar,voutMult_ajustar);
      v_out <= resize(voutMult_ajustar*VOUTSCALE, v_out);

      Iin <= to_slv(il);
      Vout <= to_slv(v_out);
```

```
end Behavioral;
```

II.1.4 MODELO EN COMA FIJA (QX.Y 16 BITS)

```
library IEEE, ieee_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;

use ieee_proposed.fixed_float_types.all;
use ieee_proposed.fixed_pkg.all;

entity BuckConverter is
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Mosfet : in  STD_LOGIC; --SW
           Vg : in  STD_LOGIC_VECTOR (16 downto 0);--Q10.6
           Ir : in  STD_LOGIC_VECTOR (16 downto 0);--Q26.-10
           Vout : out  STD_LOGIC_VECTOR (16 downto 0);--Q10.6
           Iin : out  STD_LOGIC_VECTOR (19 downto 0));--Q9.10
end BuckConverter;

architecture Behavioral of BuckConverter is

 constant dt : real := 10.0e-9;
 constant L : real := 0.002;--2mH
 constant C : real := 0.000035;

 constant dtL : sfixed(-13 downto -30) := to_sfixed(dt/L, -13, -30);--valor para ser multiplicado a iL* para obtener iL
 constant VOUTSCALE : sfixed(-23 downto -40) := to_sfixed((dt*dt)/(C*L),-23,-40);--valor para ser multiplicado a Vout* para obtener Vout
 constant il_inicial : sfixed(9 downto -10) := to_sfixed(0.0, 9, -10);--valor de iL* en caso de Reset=1
 constant vout_inicial : sfixed(10 downto -6) := to_sfixed(0.0, 10, -6);--valor de Vout* en caso de Reset=1
 signal v_out : sfixed (10 downto -6);--señal auxiliar de Vout para trabajar en sfixed
 signal vout_as : sfixed (39 downto 10);--valor de Vout* nada más salir del sumador, sin estar ajustado a Q38.-10
 signal Vg_aux : sfixed (10 downto -6);--señal auxiliar de Vg para trabajar en sfixed

 signal il : sfixed (9 downto -10); --señal auxiliar de Iin para tranajar en sfixed
 signal il_as : sfixed (26 downto -6);--valor de iL* nada más salir del sumador, sin estar ajustado a Q25.6
 signal Ir_aux : sfixed (26 downto 10);--señal auxiliar de Ir para trabajar en sfixed
 signal il_aux : sfixed (10 downto -6);--ajustamos Vg-Vout a Q10.6
 signal il_ajustar : sfixed (25 downto -6);--ajustamos el valor de iL* para entrar al FF a Q25.6
```

```
      signal ilMult_ajustar : sfixed (25 downto 8);--ajustamos iL* a
Q25.-8 para que tenga 18 bits y poder pasarlo al multiplicador
      signal ilas_ajustar : sfixed (26 downto 10);--ajustamos iL* a
Q26.-10 para poder restarle iR
      signal vout_aux : sfixed (26 downto 10);--ajustamos iL*-iR a
Q26.-10
      signal vout_ajustar : sfixed (38 downto 10);--ajustamos el valor
de Vout* para entrar al FF a Q38.-13
      signal voutMult_ajustar : sfixed (38 downto 21);--ajustamos
Vout* a Q38.-21 para que tenga 18 bits y poder pasarlo al
multiplicador


    begin

      Vg_aux <= to_sfixed (Vg,Vg_aux);--pasamos Vg de std_logic a
sfixed para poder operar con el resto de señales
      Ir_aux <= to_sfixed (Ir,Ir_aux);--pasamos Ir de std_logic a
sfixed para poder operar con el resto de señales

      process(Mosfet,Vg_aux,v_out,il,il_as,Ir_aux,il_ajustar,ilas_ajus
tar)
      begin
            if Mosfet = '1' then --interruptor cerrado(SW=1)
                  il_aux <= resize(Vg_aux-v_out,il_aux);
                  ilas_ajustar <= resize(il_ajustar,ilas_ajustar);
                  vout_aux <= resize(ilas_ajustar-Ir_aux,vout_aux);

            else
                  if il <= 0.0 then
                        il_aux <= to_sfixed(0.0, il_aux);
                        vout_aux <= resize(-Ir_aux,vout_aux);

                  else
                        il_aux <= resize(-v_out,il_aux);
                        ilas_ajustar <=resize(il_ajustar,ilas_ajustar);
                        vout_aux<=resize(ilas_ajustar-Ir_aux,vout_aux);

                  end if;
            end if;
      end process;

      process(Clk, Reset)
      begin
          if Reset = '1' then
            il_ajustar <= resize(il_inicial,il_ajustar);
            vout_ajustar <= resize(vout_inicial,vout_ajustar);

          elsif rising_edge(Clk) then
            il_ajustar <= resize(il_ajustar + il_aux,il_ajustar);
            vout_ajustar<=resize(vout_ajustar+vout_aux,vout_ajustar);


            end if;
      end process;

      --sacar iL=(iL*)*(dt/L)
      ilMult_ajustar <= resize(il_ajustar,ilMult_ajustar);
      il <= resize(ilMult_ajustar*dtL, il);

      --sacar Vout=(Vout*)*(dt/L)*(dt/C)
```

```
 voutMult_ajustar <= resize(vout_ajustar,voutMult_ajustar);
 v_out <= resize(voutMult_ajustar*VOUTSCALE, v_out);

 Iin <= to_slv(il);
 Vout <= to_slv(v_out);

end Behavioral;
```

II.2 MODELOS DEL ADC

II.2.1 MODELO REAL

```
library IEEE, work;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;
use WORK.MyConvPack.all; -- Function floor

entity ADC is
    Port ( Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           entradaAnalog : in real;
           start : in STD_LOGIC;
           salidaDig : out  std_logic_vector(19 downto 0));

end ADC;


architecture Behavioral of ADC is

 constant constante : real := 255.0; --8bits
 constant const_cero : real := 0.0;
 constant Vref : real := 3.3;

 signal startR : STD_LOGIC;
 signal startADC : STD_LOGIC;
 signal muestra : real;
 signal salidaDig_aux : real := 0.0;


begin

 process(Clk, Reset)
 begin
       if Reset = '1' then
             startADC <= '0';

       elsif rising_edge(Clk) then
             if start= '1' and startR = '0' then
                   startADC <= '1';
             else
                   startADC <= '0';

             end if;
```

```
            startR <= start;
      end if;
 end process;

 process
 begin

      wait until startADC = '1';
      muestra <= entradaAnalog;

      wait for 10 us;

      salidaDig_aux <= muestra;

      wait for 1 us;

      if salidaDig_aux < const_cero then
            salidaDig_aux <= const_cero;
      end if;

 end process;

 salidaDig <= conv_std_logic_vector(floor(salidaDig_aux), 20);


end Behavioral;
```

II.2.2 MODELO EN COMA FIJA

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity AdcStdVector2StdVector is
 port(
      Clk : in std_logic;
      Reset : in std_logic;
      Start : in std_logic;
      entradaAnalog : in std_logic_vector(19 downto 0);
      salidaDig : out std_logic_vector(19 downto 0)
      );
end AdcStdVector2StdVector;

architecture Behavioral of AdcStdVector2StdVector is

 signal sample : std_logic_vector(19 downto 0);

 signal counter : integer range 0 to 110; -- 11 us / 10 ns = 110

 signal startR, startR2 : std_logic;

begin

 prStart : process(Reset, Clk)
 begin
      if Reset = '1' then
```

```
            startR <= '0';
            startR2 <= '0';
      elsif rising_edge(Clk) then
            startR <= Start;
            startR2 <= StartR;
      end if;
 end process prStart;

 prDelay : process(Reset, Clk)
 begin
      if Reset = '1' then
            counter <= 0;
            sample <= (others => '0');
            salidaDig <= (others => '0');
      elsif rising_edge(Clk) then
            if StartR = '1' and startR2 = '0' then
                  counter <= 1;
                  sample <= entradaAnalog;
            elsif counter = 110 then
                  salidaDig <= sample;
            elsif counter /= 0 then
                  counter <= counter + 1;
            end if;
      end if;
 end process prDelay;

end Behavioral;
```

II.3 MODELO DEL REGULADOR

```
library WORK, IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_signed.all; -- conv_integer
  use IEEE.std_logic_arith.all; -- conv_std_logic_vector


entity RegulatorQXY is
  port(
        Clk : in std_logic;
        Reset : in std_logic;
        Enable : in std_logic;
        CurrentRef : in std_logic_vector(19 downto 0); -- Q19.0
        CurrentMeasured : in  std_logic_vector(19 downto 0); -- Q19.0
        DutyCycle : out std_logic_vector(14 downto 0) -- Q14.0;
  );
end RegulatorQXY;


architecture Behavioral of RegulatorQXY is

  -- Beta 6
  constant Y0Min : std_logic_vector(36 downto 0) := conv_std_logic_vector(0,37);
  constant Y0Max : std_logic_vector(36 downto 0) :=
  "0001111101011100001010001100000000000"; -- Q2.34 980 aprox Beta 6



  signal Y0 : std_logic_vector(34 downto 0); -- Q1.33 Beta 6
  signal Y1 : std_logic_vector(34 downto 0); -- Q1.33 Beta 6
  signal Y0Aux : std_logic_vector(35 downto 0); -- Q2.33 Beta 6

  constant ARapido : std_logic_vector(1 downto 0) := "01"; -- Q-11.12   Beta 6
  constant BRapido : std_logic_vector(1 downto 0) := "01"; -- Q-12.13   Beta 6
```

```
  signal AX0Rapido : std_logic_vector(21 downto 0); -- Q-1.22 Beta 6
  signal BX1Rapido : std_logic_vector(21 downto 0); -- Q-2.23 Beta 6

  signal XsRapido : std_logic_vector(23 downto 0); -- Q0.23 Beta 6

  signal errormA : std_logic_vector(19 downto 0); -- Q19.0 mA
  signal X0 : std_logic_vector(19 downto 0); -- Q9.10 A
  signal X1 : std_logic_vector(19 downto 0); -- Q9.10 A

  constant SCALEDUTY : std_logic_vector(13 downto 0) := "01001110001000"; -- Q13.0

  signal dutyCycleAux : std_logic_vector(31 downto 0); -- Q14.17;

  -- Q19.0 mA. Beta 6
  signal currentRefMenos25p, currentRefMas25p : std_logic_vector(19 downto 0);

  signal LazoLento : std_logic; -- Beta 6
  signal CambioConsignaGrande : std_logic; -- Beta 6
  signal CurrentRefR : std_logic_vector(19 downto 0); -- Beta 6


begin

  errormA <= CurrentRef - CurrentMeasured;

  REGISTROS: process (Clk, Reset)
  begin
        if Reset = '1' then
              Y1 <= (others => '0');
              X0 <= (others => '0');
              X1 <= (others => '0');
              CurrentRefR <= (others => '0');

        elsif rising_edge(clk) then
              if Enable = '1' then
                    Y1 <= Y0;
                    X0 <= errormA; -- Q9.10.
                    X1 <= X0;
                    CurrentRefR <= CurrentRef;
              end if;
        end if;
  end process REGISTROS;

  calculo: process(Clk, reset)
  begin
        if Reset = '1' then
              Y0aux  <= (others => '0');

        elsif rising_edge(Clk) then
              -- (Q0.23 que pasa a Q0.33) + Q1.33 = Q2.33
              Y0aux <= (XsRapido(23) & XsRapido & "0000000000") + (Y1(34) & Y1);
        end if;
  end process calculo;

  AX0Rapido <= ARapido * X0; -- Q-11.12 * Q9.10 = Q-1.22
  BX1Rapido <= BRapido * X1; -- Q-12.13 * Q9.10 = Q-2.23

  -- Q-1.22 - Q-2.23 = Q0.23
  XsRapido <= (AX0Rapido(21) & AX0Rapido & "0") - (BX1Rapido(21) & BX1Rapido);

  Y0 <= Y0MAX(34 downto 0) when Y0aux>Y0MAX else -- Beta 6
        Y0MIN(34 downto 0) when Y0aux<Y0MIN else -- Beta 6
        Y0aux(34 downto 0); --Q1.33  Beta 6
  -- Q13.0 * Q0.17 = Q14.17 Beta 6
  dutyCycleAux <= SCALEDUTY * ( '0' & Y0(32 downto 16));
  DutyCycle <= dutyCycleAux(31 downto 17);


end Behavioral;
```

II.4 FICHERO TOP

II.4.1 MODELO REAL

```
library IEEE, work;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;
use WORK.MyConvPack.all; -- Function floor

entity Top is
 port (
       Clk : in std_logic;
       Reset : in std_logic;
       Ir : in real;
       Vout : out real;
       Iref : in std_logic_vector(19 downto 0)
 );
end Top;

architecture Behavioral of Top is

 component BuckConverter
 port(
       --In
       Clk : in std_logic;
       Reset : in std_logic;
       Mosfet : in std_logic;
       --Interruptor(SW) -- On = '1', off = '0'
       Vg : in real;
       Ir : in real;
       -- Out
       Vout : out real;
       Iin : out real

 );
 end component;

 component ADC
 port(
       Clk : in  STD_LOGIC;
       Reset : in  STD_LOGIC;
       entradaAnalog : in real;
       start : in STD_LOGIC;
       salidaDig : out  std_logic_vector(19 downto 0)
 );
 end component;

 component RegulatorQXY
 port(
       Clk : in std_logic;
       Reset : in std_logic;
       Enable : in std_logic;
       CurrentRef : in std_logic_vector(19 downto 0); -- Q19.0
       CurrentMeasured : in std_logic_vector(19 downto 0);-- Q19.0
       DutyCycle : out std_logic_vector(14 downto 0) -- Q14.0;
 );
```

```
      end component;


      signal pwmOutput : std_logic;
      signal vg  : real := 0.0;
      signal cntEnableReg : std_logic_vector(13 downto 0);
      constant  cntEnableRegMAX  :  std_logic_vector(13  downto  0)  :=
conv_std_logic_vector(5000, cntEnableReg'length);
      signal enableReg : std_logic;

      signal tensionShunt : real;
      signal currentADC,currentADCReg : std_logic_vector(19 downto 0);

      signal Iin_aux: real;
      signal contadorPWM : std_logic_vector(12 downto 0);
      signal DutyPWMReal : real;
      signal DutyPWMReg : std_logic_vector(14 downto 0);
      signal DutyPWMFinal : std_logic_vector(12 downto 0);
      constant   DUTYPWM   :   std_logic_vector(12   downto   0)   :=
conv_std_logic_vector(750,13);
      constant VGPEAKVALUE : real :=
     325.26911934581186122438840656823; -- 230 * sqr(2)
     -- 2*pi*50
      constant W : real := 314.15926535897932384626433832795;
      constant pi2_3 : real := 2.09439510239319549230842892218 63;
      constant CLKVGPERIOD : time := 10 ns;
      constant CLKVGPERIODAT : real := 10.0e-9;
      signal vg1, vg2, vg3 : real := 0.0; -- Fases de la entrada
      signal vgplus, vgminus : real := 0.0;
      signal startADC : std_logic;


     BEGIN

      -- Generación de la tensión trifásica rectificada
      v3gGen: process
            variable t : real := 0.0;
      begin
            vg1 <= VGPEAKVALUE*sin(W*t);
            vg2 <= VGPEAKVALUE*sin(W*t+pi2_3);
            vg3 <= VGPEAKVALUE*sin(W*t-pi2_3);
            t:=t+CLKVGPERIODAT;
            wait for CLKVGPERIOD;
      end process v3gGen;

      -- Valor rectificado de la tensión trifásica de entrada
      vg <=       vgplus - vgminus;
      vgplus <= vg1 when ((vg1 > vg2) and (vg1 > vg3)) else
                        vg2 when (vg2 > vg3) else
                        vg3;

      vgminus <= vg1 when ((vg1 < vg2) and (vg1 < vg3)) else
                        vg2 when (vg2 < vg3) else
                        vg3;


      process(Clk, Reset)
      begin
            if Reset = '1' then
                  contadorPWM <= (others => '0');
            elsif rising_edge(Clk) then
```

```
            if contadorPWM < 4999 then
                  contadorPWM <= contadorPWM + 1;
            else
                  contadorPWM <= (others => '0');
            end if;
      end if;
end process;

process(Clk, Reset)
begin
      if Reset = '1' then
            pwmOutput <= '0';
      elsif rising_edge(Clk) then
            if contadorPWM < DutyPWMFinal then
                  pwmOutput <= '1';
            else
                  pwmOutput <= '0';
            end if;
      end if;
end process;

--Buck converter
buck: BuckConverter
port map (
      Clk => Clk, --Componente => propio fichero
      Reset => Reset,
      Mosfet => pwmOutput,
      Vg => vg,
      Ir => Ir,
      Iin => Iin_aux,
      Vout => Vout
);

--ADC
adc_top: ADC
port map (
      Clk => Clk,
      Reset => Reset,
      entradaAnalog => tensionShunt,
      start => startADC,
      salidaDig => currentADC
);

process(Clk, Reset)
begin
      if Reset = '1' then
            tensionShunt <= 0.0;
      elsif rising_edge(Clk) then
            tensionShunt <= Iin_aux * 1024.0;
      end if;
end process;

regulator : RegulatorQXY
port map(
      Clk => Clk,
      Reset => Reset,
      Enable => enableReg,
      CurrentRef => Iref,
      CurrentMeasured => currentADC,
      DutyCycle => DutyPWMReg
);
```

```
--DutyPWMFinal <= DutyPWMReg(12 downto 0); --Lazo cerrado
DutyPWMFinal <= conv_std_logic_vector(1500,13); --Lazo abierto

currentADCReg <= currentADC;

process (clk, reset)
begin
      if reset = '1' then
            cntEnableReg <= (others => '0');

      elsif rising_edge(clk) then
            if cntEnableReg < cntEnableRegMAX-1 then
                  cntEnableReg <= cntEnableReg + 1;
            else
                  cntEnableReg <= (others => '0');
            end if;
      end if;
end process;

process (clk, reset)
begin
      if reset = '1' then
            enableReg <= '0';

      elsif rising_edge(clk) then
            if cntEnableReg = cntEnableRegMAX-5 then
                  enableReg <= '1';
            else
                  enableReg <= '0';
            end if;
      end if;
end process;

process (clk, reset)
begin
      if reset = '1' then
            startADC <= '0';

      elsif rising_edge(clk) then
            if cntEnableReg = cntEnableRegMAX-2500 then
                  startADC <= '1';
            else
                  startADC <= '0';
            end if;
      end if;
end process;

end Behavioral;
```

II.4.2 MODELO EN COMA FIJA (QX.Y 6 BITS)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
```

```
use IEEE.std_logic_arith.all;

entity Top is
 port (
       Clk : in std_logic;
       Reset : in std_logic;
       Ir : in std_logic_vector(6 downto 0);
       Vout : out std_logic_vector(6 downto 0);
       Iref : in std_logic_vector(19 downto 0)
 );
end Top;

architecture Behavioral of Top is

 component BuckConverter
 port(
       Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Mosfet : in  STD_LOGIC; --SW
               Vg : in  STD_LOGIC_VECTOR (6 downto 0);--Q10.-4
           Ir : in  STD_LOGIC_VECTOR (6 downto 0);--Q26.-20
           Vout : out  STD_LOGIC_VECTOR (6 downto 0);--Q10.-4
           Iin : out  STD_LOGIC_VECTOR (19 downto 0)--Q9.10
 );
 end component;

 component AdcStdVector2StdVector
 port(
       Clk : in std_logic;
       Reset : in std_logic;
       Start : in std_logic;
       entradaAnalog : in std_logic_vector(19 downto 0);
       salidaDig : out std_logic_vector(19 downto 0)
 );
 end component;


 component RegulatorQXY
 port(
       Clk : in std_logic;
       Reset : in std_logic;
       Enable : in std_logic;
       CurrentRef : in std_logic_vector(19 downto 0); -- Q19.0
       CurrentMeasured  :  in    std_logic_vector(19  downto  0);  --
Q19.0
       DutyCycle : out std_logic_vector(14 downto 0) -- Q14.0;
 );
 end component;


 component VgPrecalc
 port(
       signal Clk : in std_logic;
       signal Reset : in std_logic;
       signal Addr : in std_logic_vector(9 downto 0);
       signal Do : out std_logic_vector(15 downto 0)
 );
 end component;


 signal pwmOutput : std_logic;
```

```
	signal vg   : std_logic_vector(15 downto 0);
	signal vgAuxConSigno : std_logic_vector(6 downto 0);

	signal cntEnableReg : std_logic_vector(13 downto 0);
	constant  cntEnableRegMAX  :  std_logic_vector(13  downto  0)  :=
conv_std_logic_vector(5000, cntEnableReg'length);
	signal enableReg : std_logic;

	signal tensionShunt : std_logic_vector(11 downto 0);
	signal  currentADC,  currentADCReg  :  std_logic_vector(19  downto
0);
	signal currentReg : std_logic_vector(19 downto 0);

	signal Iin_aux : std_logic_vector(19 downto 0);
	signal contadorPWM : std_logic_vector(12 downto 0);
	signal DutyPWMReg : std_logic_vector(14 downto 0); --Q14.0
	signal DutyPWMFinal : std_logic_vector(12 downto 0);
	constant   DUTYPWM   :   std_logic_vector(12   downto   0)   :=
conv_std_logic_vector(750,13);

	--señales auxiliares que habrá que borrar después
	signal startADC : std_logic;
	signal addrVgMem : std_logic_vector(9 downto 0);
	-- Address of Vg memory
	signal counterVgMem     : std_logic_vector (9 downto 0);
	-- Counter to address Vg memory

    BEGIN

	process(Clk, Reset)
	begin
		if Reset = '1' then
			contadorPWM <= (others => '0');
		elsif rising_edge(Clk) then
			if contadorPWM < 4999 then
				contadorPWM <= contadorPWM + 1;
			else
				contadorPWM <= (others => '0');
			end if;
		end if;
	end process;

	process(Clk, Reset)
	begin
		if Reset = '1' then
			pwmOutput <= '0';
		elsif rising_edge(Clk) then
			if contadorPWM < DutyPWMFinal then
				pwmOutput <= '1';
			else
				pwmOutput <= '0';
			end if;
		end if;
	end process;

		vgAuxConSigno <= '0' & vg(15 downto 10);

	--Buck converter
	buck: BuckConverter
	port map (
		Clk => Clk,	Reset => Reset,
```

```
        Mosfet => pwmOutput,
        Vg => vgAuxConSigno,
        Ir => Ir,
        Iin => Iin_aux,
        Vout => Vout
);

--ADC
adc_top: AdcStdVector2StdVector
port map (
        Clk => Clk,
        Reset => Reset,
        entradaAnalog => Iin_aux,
        start => startADC,
        salidaDig => currentADC
);

-- Regulador
regulator : RegulatorQXY
port map(
        Clk => Clk,
        Reset => Reset,
        Enable => enableReg,
        CurrentRef => Iref,
        CurrentMeasured => currentReg,
        DutyCycle => DutyPWMReg
);

currentReg <= currentADC;
--DutyPWMFinal <= DutyPWMReg(12 downto 0);
DutyPWMFinal <= conv_std_logic_vector(1500,13);


process (clk, reset)
begin
        if reset = '1' then
                cntEnableReg <= (others => '0');
        elsif rising_edge(clk) then
                if cntEnableReg < cntEnableRegMAX-1 then
                        cntEnableReg <= cntEnableReg + 1;
                else
                        cntEnableReg <= (others => '0');
                end if;
        end if;
end process;

process (clk, reset)
begin
        if reset = '1' then
                enableReg <= '0';
        elsif rising_edge(clk) then
                if cntEnableReg = cntEnableRegMAX-5 then
                        enableReg <= '1';
                else
                        enableReg <= '0';
                end if;
        end if;
end process;


process (clk, reset)
```

```
begin
	if reset = '1' then
		startADC <= '0';
	elsif rising_edge(clk) then
		if cntEnableReg = cntEnableRegMAX-2500 then
			startADC <= '1';
		else
			startADC <= '0';
		end if;
	end if;
end process;

-- Vg precalculated memory
vgMem: VgPrecalc
port map(
	Clk => Clk,
	Reset => Reset,
	Addr => addrVgMem,
	Do => vg
);

auxVgAddressing: process (Clk, Reset)
begin
	if Reset = '1' then
		counterVgMem <= (others => '0');
	elsif rising_edge(Clk) then
		if counterVgMem < x"3E7" then
		-- Divides the clock freq by 1000
			counterVgMem <= counterVgMem + 1;
		else
			counterVgMem <= (others => '0');
		end if;
	end if;
end process auxVgAddressing;
vgAddressing: process(Clk, Reset)
begin
	if Reset = '1' then
		addrVgMem <= (others => '0');
	elsif rising_edge(Clk) then
		if counterVgMem = x"3E6" then
	-- Every 1000 Clk cycles, addrVgMem is incremented by 1
			if addrVgMem < x"3E7" then
				addrVgMem <= addrVgMem + 1;
			else
			     -- When counterVgMem reaches value "1000"
			(Vg semicycle), it resets
				addrVgMem <= (others => '0');
			end if;
		end if;
	end if;
end process vgAddressing;


end Behavioral;
```

II.4.3 MODELO EN COMA FIJA (QX.Y 12 BITS)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;

entity Top is
 port (
       Clk : in std_logic;
       Reset : in std_logic;
       Ir : in std_logic_vector(12 downto 0);--Q26.-14
       Vout : out std_logic_vector(12 downto 0);--Q10.2
       Iref : in std_logic_vector(19 downto 0)
 );
end Top;

architecture Behavioral of Top is

 component BuckConverter
 port(
       Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Mosfet : in  STD_LOGIC; --SW
               Vg : in  STD_LOGIC_VECTOR (12 downto 0);--Q10.2
           Ir : in  STD_LOGIC_VECTOR (12 downto 0);--Q26.-14
           Vout : out  STD_LOGIC_VECTOR (12 downto 0);--Q10.2
           Iin : out  STD_LOGIC_VECTOR (19 downto 0)--Q9.10
 );
 end component;

 component AdcStdVector2StdVector
 port(
       Clk : in std_logic;
       Reset : in std_logic;
       Start : in std_logic;
       entradaAnalog : in std_logic_vector(19 downto 0);
       salidaDig : out std_logic_vector(19 downto 0)
 );
 end component;


 component RegulatorQXY
 port(
       Clk : in std_logic;
       Reset : in std_logic;
       Enable : in std_logic;
       CurrentRef : in std_logic_vector(19 downto 0); -- Q19.0
       CurrentMeasured : in std_logic_vector(19 downto 0);-- Q19.0
       DutyCycle : out std_logic_vector(14 downto 0) -- Q14.0;
 );
 end component;


 component VgPrecalc
 port(
       signal Clk : in std_logic;
       signal Reset : in std_logic;
```

```
            signal Addr : in std_logic_vector(9 downto 0);
            signal Do : out std_logic_vector(15 downto 0)
      );
      end component;


      signal pwmOutput : std_logic;
      signal vg  : std_logic_vector(15 downto 0);
      signal vgAuxConSigno : std_logic_vector(12 downto 0);

      signal cntEnableReg : std_logic_vector(13 downto 0);
      constant  cntEnableRegMAX  :  std_logic_vector(13  downto  0)  :=
conv_std_logic_vector(5000, cntEnableReg'length);
      signal enableReg : std_logic;

      signal tensionShunt : std_logic_vector(11 downto 0);
      signal  currentADC,  currentADCReg  :  std_logic_vector(19  downto
0);
      signal currentReg : std_logic_vector(19 downto 0);

      signal Iin_aux : std_logic_vector(19 downto 0);

      signal contadorPWM : std_logic_vector(12 downto 0);
      signal DutyPWMReg : std_logic_vector(14 downto 0); --Q14.0
      signal DutyPWMFinal : std_logic_vector(12 downto 0);
      constant   DUTYPWM   :   std_logic_vector(12   downto   0)   :=
conv_std_logic_vector(750,13);
      signal startADC : std_logic;
      signal addrVgMem : std_logic_vector(9 downto 0);
      signal counterVgMem     : std_logic_vector (9 downto 0);

    BEGIN

      process(Clk, Reset)
      begin
            if Reset = '1' then
                  contadorPWM <= (others => '0');
            elsif rising_edge(Clk) then
                  if contadorPWM < 4999 then
                        contadorPWM <= contadorPWM + 1;
                  else
                        contadorPWM <= (others => '0');
                  end if;
            end if;
      end process;

      process(Clk, Reset)
      begin
            if Reset = '1' then
                  pwmOutput <= '0';
            elsif rising_edge(Clk) then
                  if contadorPWM < DutyPWMFinal then
      pwmOutput <= '1';
                  else
                        pwmOutput <= '0';
                  end if;
            end if;
      end process;
```

```
	vgAuxConSigno <= '0' & vg(15 downto 4); -- La memoria tiene 16
bits de ancho, pero nos interesan 13 ya que es un Q10.2 (10 + 2 + 1 de
signo).

	--Buck converter
	buck: BuckConverter
	port map (
		Clk => Clk,
		Reset => Reset,
		Mosfet => pwmOutput,
		Vg => vgAuxConSigno,
		Ir => Ir,
		Iin => Iin_aux,
		Vout => Vout
	);


	--ADC
	adc_top: AdcStdVector2StdVector
	port map (
		Clk => Clk, --Componente => propio fichero
		Reset => Reset,
		entradaAnalog => Iin_aux,
		start => startADC,
		salidaDig => currentADC
	);

	-- Regulador
	regulator : RegulatorQXY -- Regulador nuevo en QX.Y
	port map(
		Clk => Clk,
		Reset => Reset,
		Enable => enableReg,
		CurrentRef => Iref,
		CurrentMeasured => currentReg,
		DutyCycle => DutyPWMReg
	);


	currentReg <= currentADC;
	DutyPWMFinal <= DutyPWMReg(12 downto 0);
	--DutyPWMFinal <= conv_std_logic_vector(1500,13);

	process (clk, reset)
	begin
		if reset = '1' then
			cntEnableReg <= (others => '0');
		elsif rising_edge(clk) then
			if cntEnableReg < cntEnableRegMAX-1 then
				cntEnableReg <= cntEnableReg + 1;
			else
				cntEnableReg <= (others => '0');
			end if;
		end if;
	end process;

	process (clk, reset)
	begin
		if reset = '1' then
			enableReg <= '0';
		elsif rising_edge(clk) then
```

```
            if cntEnableReg = cntEnableRegMAX-5 then
                  enableReg <= '1';
            else
                  enableReg <= '0';
            end if;
      end if;
end process;


process (clk, reset)
begin
      if reset = '1' then
            startADC <= '0';
      elsif rising_edge(clk) then
            if cntEnableReg = cntEnableRegMAX-2500 then
                  startADC <= '1';
            else
                  startADC <= '0';
            end if;
      end if;
end process;

vgMem: VgPrecalc
port map(
      Clk => Clk,
      Reset => Reset,
      Addr => addrVgMem,
      Do => vg
);


auxVgAddressing: process (Clk, Reset)
begin
      if Reset = '1' then
            counterVgMem <= (others => '0');
      elsif rising_edge(Clk) then
            if counterVgMem < x"3E7" then
            -- Divides the clock freq by 1000
                  counterVgMem <= counterVgMem + 1;
            else
                  counterVgMem <= (others => '0');
            end if;
      end if;
end process auxVgAddressing;

vgAddressing: process(Clk, Reset)
begin
      if Reset = '1' then
            addrVgMem <= (others => '0');
      elsif rising_edge(Clk) then
            if counterVgMem = x"3E6" then
            -- Every 1000 Clk cycles, addrVgMem is incremented by
      1
                  if addrVgMem < x"3E7" then
                        addrVgMem <= addrVgMem + 1;
                  else
                       -- When counterVgMem reaches value "1000"
                  (Vg semicycle), it resets
                        addrVgMem <= (others => '0');
                  end if;
            end if;
```

```
        end if;
    end process vgAddressing;


    end Behavioral;
```

II.4.4 MODELO EN COMA FIJA (QX.Y 16 BITS)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;
use IEEE.std_logic_arith.all;

entity Top is
 port (
       Clk : in std_logic;
       Reset : in std_logic;
       Ir : in std_logic_vector(16 downto 0);
       Vout : out std_logic_vector(16 downto 0);
       Iref : in std_logic_vector(19 downto 0)
 );
end Top;

architecture Behavioral of Top is

 component BuckConverter
 port(
       Clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Mosfet : in  STD_LOGIC; --SW
               Vg : in  STD_LOGIC_VECTOR (16 downto 0);--Q10.6
           Ir : in  STD_LOGIC_VECTOR (16 downto 0);--Q26.-10
           Vout : out  STD_LOGIC_VECTOR (16 downto 0);--Q10.6
           Iin : out  STD_LOGIC_VECTOR (19 downto 0)--Q9.10
 );
 end component;

 component AdcStdVector2StdVector
 port(
       Clk : in std_logic;
       Reset : in std_logic;
       Start : in std_logic;
       entradaAnalog : in std_logic_vector(19 downto 0);
       salidaDig : out std_logic_vector(19 downto 0)
 );
 end component;


 component RegulatorQXY
 port(
       Clk : in std_logic;
       Reset : in std_logic;
       Enable : in std_logic;
       CurrentRef : in std_logic_vector(19 downto 0); -- Q19.0
       CurrentMeasured  :  in    std_logic_vector(19  downto  0);  --
Q19.0
```

```
            DutyCycle : out std_logic_vector(14 downto 0) -- Q14.0;
      );
      end component;


      component VgPrecalc
      port(
            signal Clk : in std_logic;
            signal Reset : in std_logic;
            signal Addr : in std_logic_vector(9 downto 0);
            signal Do : out std_logic_vector(15 downto 0)
      );
      end component;


      signal pwmOutput : std_logic;
      signal vg   : std_logic_vector(15 downto 0);
      signal vgAuxConSigno : std_logic_vector(16 downto 0);

      signal cntEnableReg : std_logic_vector(13 downto 0);
      constant  cntEnableRegMAX  :  std_logic_vector(13  downto  0)  :=
conv_std_logic_vector(5000, cntEnableReg'length);
      signal enableReg : std_logic;

      signal tensionShunt : std_logic_vector(11 downto 0);
      signal  currentADC,  currentADCReg  :  std_logic_vector(19  downto
0);
      signal currentReg : std_logic_vector(19 downto 0);

      signal Iin_aux : std_logic_vector(19 downto 0);

      signal contadorPWM : std_logic_vector(12 downto 0);
      signal DutyPWMReg : std_logic_vector(14 downto 0); --Q14.0
      signal DutyPWMFinal : std_logic_vector(12 downto 0);
      constant   DUTYPWM   :   std_logic_vector(12   downto   0)   :=
conv_std_logic_vector(750,13);

      signal startADC : std_logic;
      signal addrVgMem : std_logic_vector(9 downto 0);
      signal counterVgMem     : std_logic_vector (9 downto 0);

     BEGIN

      process(Clk, Reset)
      begin
            if Reset = '1' then
                  contadorPWM <= (others => '0');
            elsif rising_edge(Clk) then
                  if contadorPWM < 4999 then
                        contadorPWM <= contadorPWM + 1;
                  else
                        contadorPWM <= (others => '0');
                  end if;
            end if;
      end process;

      process(Clk, Reset)
      begin
            if Reset = '1' then
                  pwmOutput <= '0';
            elsif rising_edge(Clk) then
```

```
            if contadorPWM < DutyPWMFinal then
                  pwmOutput <= '1';
            else
                  pwmOutput <= '0';
            end if;
      end if;
end process;

vgAuxConSigno <= '0' & vg;
     -- La memoria tiene 16 bits de ancho, pero nos interesan 17
ya que es un Q10.6 (10 + 6 + 1 de signo).


--Buck converter
buck: BuckConverter
port map (
      Clk => Clk,       Reset => Reset,
      Mosfet => pwmOutput,
      Vg => vgAuxConSigno,
      Ir => Ir,
      Iin => Iin_aux,
      Vout => Vout
);


--ADC
adc_top: AdcStdVector2StdVector
port map (
      Clk => Clk,
      Reset => Reset,
      entradaAnalog => Iin_aux,
      start => startADC,
      salidaDig => currentADC
);

-- Regulador
regulator : RegulatorQXY
port map(
      Clk => Clk,
      Reset => Reset,
      Enable => enableReg,
      CurrentRef => Iref,
      CurrentMeasured => currentReg,
      DutyCycle => DutyPWMReg
);


currentReg <= currentADC;
--DutyPWMFinal <= DutyPWMReg(12 downto 0);
DutyPWMFinal <= conv_std_logic_vector(2500,13);

process (clk, reset)
begin
      if reset = '1' then
            cntEnableReg <= (others => '0');
      elsif rising_edge(clk) then
            if cntEnableReg < cntEnableRegMAX-1 then
                  cntEnableReg <= cntEnableReg + 1;
            else
                  cntEnableReg <= (others => '0');
            end if;
```

```
        end if;
    end process;

    process (clk, reset)
    begin
        if reset = '1' then
            enableReg <= '0';
        elsif rising_edge(clk) then
            if cntEnableReg = cntEnableRegMAX-5 then
                enableReg <= '1';
            else
                enableReg <= '0';
            end if;
        end if;
    end process;
    process (clk, reset)
    begin
        if reset = '1' then
            startADC <= '0';
        elsif rising_edge(clk) then
            if cntEnableReg = cntEnableRegMAX-2500 then
                startADC <= '1';
            else
                startADC <= '0';
            end if;
        end if;
    end process;

    vgMem: VgPrecalc
    port map(
        Clk => Clk,
        Reset => Reset,
        Addr => addrVgMem,
        Do => vg
    );


    auxVgAddressing: process (Clk, Reset)
    begin
        if Reset = '1' then
            counterVgMem <= (others => '0');
        elsif rising_edge(Clk) then
            if counterVgMem < x"3E7" then
            -- Divides the clock freq by 1000
                counterVgMem <= counterVgMem + 1;
            else
                counterVgMem <= (others => '0');
            end if;
        end if;
    end process auxVgAddressing;

    vgAddressing: process(Clk, Reset)
    begin
        if Reset = '1' then
            addrVgMem <= (others => '0');
        elsif rising_edge(Clk) then
            if counterVgMem = x"3E6" then
            -- Every 1000 Clk cycles, addrVgMem is incremented by
        1
                if addrVgMem < x"3E7" then
                    addrVgMem <= addrVgMem + 1;
```

```
                    else
                        -- When counterVgMem reaches value "1000"
                    (Vg semicycle), it resets
                          addrVgMem <= (others => '0');
                    end if;
              end if;
        end if;
 end process vgAddressing;


 end Behavioral;
```

II.5 FICHERO TOP_TB

II.5.1 MODELO REAL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity TopTb is
end TopTb;

architecture Behavioral of TopTb is

 component Top
       port(
             Clk : in std_logic;
             Reset : in std_logic;
             Ir : in real;
             Vout : out real;
             Iref : in std_logic_vector(19 downto 0)
       );
 end component;


 signal clk : std_logic := '0';
 signal clkBoost : std_logic := '0';
 signal reset : std_logic := '0';
 signal vout : real;

 signal iin : std_logic_vector(11 downto 0) := (others => '0');

 constant CLKPERIOD : time := 10 ns;
 constant CLKBOOSTPERIOD : time := 10 ns;

 signal iR : real;
 constant R : real := 25.0;
 signal iRef : std_logic_vector(19 downto 0);


begin

 uut: Top port map (
       Clk => clk,
       Reset => reset,
       Ir => ir,
```

```
        Vout => Vout,
        iRef => iRef
    );

    -- 5 A => 5000 mA y aprox 5*1024 = 5000 (5120)
    iRef <= "00000001010000000000";

    -- Modelado de una batería
    iR <= (vout-216.0) / 1.0;
        -- Suponemos que las resistencias en serie de las baterías
    es de 1 Ohm, y la tensión en serie es 216.0.

    -- Clock of the regulator
    clkGen: process
    begin
        Clk <= '0';
        wait for CLKPERIOD/2;
        Clk <= '1';
        wait for CLKPERIOD/2;
    end process clkGen;

    -- Clock of the boost model
    clkBoostGen: process
    begin
        ClkBoost <= '0';
        wait for CLKBOOSTPERIOD/2;
        ClkBoost <= '1';
        wait for CLKBOOSTPERIOD/2;
    end process clkBoostGen;

    stimProc: process
    begin
        reset <= '1';

        wait for 1 ps;
        reset <= '0';
        wait for 30 ms;
        wait;
    end process stimProc;

end Behavioral;
```

II.5.2 MODELO EN COMA FIJA (QX.Y 6 BITS)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity TopTb is
end TopTb;

architecture Behavioral of TopTb is

 component Top
        port(
                Clk : in std_logic;
                Reset : in std_logic;
```

```
            Ir : in std_logic_vector(6 downto 0);
            Vout : out std_logic_vector(6 downto 0);
            Iref : in std_logic_vector(19 downto 0)
      );
end component;


signal clk : std_logic := '0';
signal clkBoost : std_logic := '0';
signal reset : std_logic := '0';
signal vout : std_logic_vector(6 downto 0);

constant CLKPERIOD : time := 10 ns;
constant CLKBOOSTPERIOD : time := 10 ns;

signal iRAux : std_logic_vector(7 downto 0); -- Q11.-4
signal iRAux2 : std_logic_vector(25 downto 0); -- Q30.-5
signal iR : std_logic_vector(6 downto 0); -- Q26.-20
signal iRef : std_logic_vector(19 downto 0);

constant   irSCALE   :   std_logic_vector(17   downto   0)   :=
"011000011010100000"; --Q18.-1
--Q10.-4 --Q10.2 "0001101100000"
constant Vbateria : std_logic_vector(6 downto 0) := "0001101";

begin

uut: Top port map (
      Clk => clk,
      Reset => reset,
      Ir => ir,
      Vout => Vout,
      iRef => iRef
);


iRef <= "00000011010000000000";
-- 13 A => 13000 mA y aprox 13*1024 = 13000 (13312)

-- Modelado de una batería
iRAux <= (vout(6) & vout) - (Vbateria(6) & Vbateria);
     --Q10.-4 - Q10.-4 =Q11.-4-- Suponemos que las resistencias
en serie de las baterías es de 1 Ohm, y la tensión en serie es
216.0.

irAux2 <= iRAux * irSCALE; -- Q11.-4 * Q18.-1 = Q30.-5

ir <= irAux2(21 downto 15);
-- Q30.-5 a Q26.-20 = Q21.15

clkGen: process
begin
      Clk <= '0';
      wait for CLKPERIOD/2;
      Clk <= '1';
      wait for CLKPERIOD/2;
end process clkGen;
clkBoostGen: process
begin
      ClkBoost <= '0';
      wait for CLKBOOSTPERIOD/2;
```

```
        ClkBoost <= '1';
        wait for CLKBOOSTPERIOD/2;
    end process clkBoostGen;

    stimProc: process
    begin
        reset <= '1';

        wait for 1 ps;
        reset <= '0';
        wait for 30 ms;
        wait;
    end process stimProc;

end Behavioral;
```

II.5.3 MODELO EN COMA FIJA (QX.Y 12 BITS)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity TopTb is
end TopTb;

architecture Behavioral of TopTb is

    component Top
        port(
            Clk : in std_logic;
            Reset : in std_logic;
            Ir : in std_logic_vector(12 downto 0);
            Vout : out std_logic_vector(12 downto 0);
            Iref : in std_logic_vector(19 downto 0)
        );
    end component;

    signal clk : std_logic := '0';
    signal clkBoost : std_logic := '0';
    signal reset : std_logic := '0';
    signal vout : std_logic_vector(12 downto 0);

    constant CLKPERIOD : time := 10 ns;
    constant CLKBOOSTPERIOD : time := 10 ns;

    signal iRAux : std_logic_vector(13 downto 0); -- Q11.2
    signal iRAux2 : std_logic_vector(31 downto 0); -- Q30.1
    signal iR : std_logic_vector(12 downto 0); -- Q26.-14 --Q26.-13

    signal iRef : std_logic_vector(19 downto 0);
    -- A: Referencia de corriente para el regulador

    constant irSCALE : std_logic_vector(17 downto 0) :=
"011000011010100000"; --Q18.-1

    constant Vbateria : std_logic_vector(12 downto 0) :=
"0001101100000";
```

```
begin


 uut: Top port map (
       Clk => clk,
       Reset => reset,
       Ir => ir,
       Vout => Vout,
       iRef => iRef
 );

 --iRef <= "00000011010000000000"; -- 13 A => 13000 mA y aprox
 13*1024 = 13000 (13312)
 iRef <= "00000111100000000000";
 -- 30 A => 30000 mA y aprox 30*1024 = 30000 (30720)
 --iRef <= "00000001010000000000"; -- 5 A => 5000 mA y aprox
 5*1024 = 5000 (5120)

 -- Modelado de una batería
 iRAux <= (vout(12) & vout) - (Vbateria(12) & Vbateria);
      --Q10.2 - Q10.2 =Q11.2 -- Suponemos que las resistencias en
 serie de las baterías es de 1 Ohm, y la tensión en serie es
 216.0.

 irAux2 <= iRAux * irSCALE; -- Q11.2 * Q18.-1 = Q30.1

 ir <= irAux2(27 downto 15);
 --Q26.-14 -- Q26.-13 ir <= irAux2(26 downto 14)
 clkGen: process
 begin
       Clk <= '0';
       wait for CLKPERIOD/2;
       Clk <= '1';
       wait for CLKPERIOD/2;
 end process clkGen;
 clkBoostGen: process
 begin
       ClkBoost <= '0';
       wait for CLKBOOSTPERIOD/2;
       ClkBoost <= '1';
       wait for CLKBOOSTPERIOD/2;
 end process clkBoostGen;

 stimProc: process
 begin
       reset <= '1';

       wait for 1 ps;
       reset <= '0';
       wait for 30 ms;
       wait;
 end process stimProc;

end Behavioral;
```

II.5.4 MODELO EN COMA FIJA (QX.Y 16 BITS)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity TopTb is
end TopTb;

architecture Behavioral of TopTb is

 component Top
       port(
             Clk : in std_logic;
             Reset : in std_logic;
             Ir : in std_logic_vector(16 downto 0);
             Vout : out std_logic_vector(16 downto 0);
             Iref : in std_logic_vector(19 downto 0)
       );
 end component;

 signal clk : std_logic := '0';
 signal clkBoost : std_logic := '0';
 signal reset : std_logic := '0';
 signal vout : std_logic_vector(16 downto 0);

 constant CLKPERIOD : time := 10 ns;
 constant CLKBOOSTPERIOD : time := 10 ns;

 signal iRAux : std_logic_vector(17 downto 0); -- Q11.6
 signal iRAux2 : std_logic_vector(35 downto 0); -- Q30.5
 signal iR : std_logic_vector(16 downto 0); -- Q26.-10

 signal iRef : std_logic_vector(19 downto 0);

 constant irSCALE : std_logic_vector(17 downto 0) :=
"011000011010100000"; --Q18.-1
 constant Vbateria : std_logic_vector(16 downto 0) :=
"00011011000000000";--Q10.6

begin


 uut: Top port map (
       Clk => clk,
       Reset => reset,
       Ir => ir,
       Vout => Vout,
       iRef => iRef
 );

 --iRef <= "00000011010000000000"; -- 13 A => 13000 mA y aprox
 13*1024 = 13000 (13312)
 iRef <= "00000111100000000000";
 -- 30 A => 30000 mA y aprox 30*1024 = 30000 (30720)
 --iRef <= "00000001010000000000"; -- 5 A => 5000 mA y aprox
 5*1024 = 5000 (5120)

 -- Modelado de una batería
```

```
	iRAux <= (vout(16) & vout) - (Vbateria(16) & Vbateria);
	     --Q10.6 - Q10.6 =Q11.6 -- Suponemos que las resistencias en
	serie de las baterías es de 1 Ohm, y la tensión en serie es
	216.0.

	irAux2 <= iRAux * irSCALE; -- Q11.6 * Q18.-1 = Q30.5

	ir <= irAux2(31 downto 15); -- Q26.-10

	clkGen: process
	begin
		Clk <= '0';
		wait for CLKPERIOD/2;
		Clk <= '1';
		wait for CLKPERIOD/2;
	end process clkGen;

	clkBoostGen: process
	begin
		ClkBoost <= '0';
		wait for CLKBOOSTPERIOD/2;
		ClkBoost <= '1';
		wait for CLKBOOSTPERIOD/2;
	end process clkBoostGen;

	stimProc: process
	begin
		reset <= '1';

		wait for 1 ps;
		reset <= '0';
		wait for 30 ms;
		wait;
	end process stimProc;

end Behavioral;
```